AD-A099 260 IBM FEDERAL SYSTEMS DIV OWEGO N Y F/G 9/2
MIL-STD-1750 CERTIFICATION STUDY.(U)
FEB 80 M L KUSHNER, D C REISIGER, W J TRACZ F33657-79-M-0858
UNCLASSIFIED IBM-6176175A NL

1 OF 3
AD
A099260

MIL-STD-1750 CERTIFICATION STUDY

CONTRACT NO.: F33657-79-M-0858

FINAL REPORT

IBM NO.: 6176175A

IBM Federal Systems Division, Owego, New York

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) 11 IBM-6176- A

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO. AD-A099260

3. RECIPIENT'S CATALOG NUMBER (1)

4. TITLE (and Subtitle)

(6) MIL-STD-175Ø CERTIFICATION STUDY. (11)

5. TYPE OF REPORT & PERIOD COVERED
FINAL REPORT.
13 Sep 1979 — 29 Feb 1980

6. PERFORMING ORG. REPORT NUMBER
6176175A

7. AUTHOR(s)
(10) M. L. Kushner
D. C. Reisiger
W. J. Tracz
L. A. White

8. CONTRACT OR GRANT NUMBER(s)
(15) F33657-79-M-Ø858

9. PERFORMING ORGANIZATION NAME AND ADDRESS
International Business Machines Corp.
Federal Systems Division
Owego, NY 13827

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
ASD/XRE
WPAFB, OH 45433

12. REPORT DATE
(11) 29 Feb 1980

13. NUMBER OF PAGES
257

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
ASD/ENASD
WPAFB, OH 45433
(12) 250

15. SECURITY CLASS. (of this report)
UNCLASSIFIED

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Instruction Set Architecture
MIL-STD-1750
Computer Architecture Verification
Computer Testing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This work presents an investigation of methods of verifying computers to an Instruction Set Architecture and recommends a method suitable for Air Force use to verify vendor produced implementations of MIL-STD-1750.

UNCLASSIFIED 401206

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

MIL-STD-1750 CERTIFICATION STUDY

CONTRACT NO.: F33657-79-M-0858

FINAL REPORT

February 29, 1980

IBM NO.: 6176175A

Approved By: J. C. Hofler, Jr.
Manager, Advanced Computer Applications

Prepared By: M. L. Kushner, Team Leader, Advanced Computer Applications
D. C. Reisiger, Advanced Computer Applications
W. J. Tracz, Software Development
L. A. White, System Test Technology

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE CF CCNTENTS

TABLE CF CONTENTS

TABLE OF CONTENTS

TABLE CF CCNTENTS

LIST OF FIGURES

LIST OF TABLES

1.0 INTRODUCTION AND SUMMARY

The Air Force's approach to solving the computer proliferation problem by specification of a standard instruction set architecture has great potential for reducing software development and maintenance costs while simultaneously capturing the benefits of technology infusion. Realization of this potential can only be achieved, however, if different implementations of MIL-STD-1750 do, in fact, precisely represent the architecture. The importance of rigorous verification that these representations are correct cannot be overstated. For these reasons, IBM considers the successful implementation of a MIL-STD-1750 certification facility to be a critically important element of the Air Force's computer architecture standardization effort. This study has been undertaken to assist in the implementation of that facility.

This study's purposes and goals follow:

- Investigate methods of verifying computers to an Instruction Set Architecture and recommend a method suitable to SEAFAC for certifying vendor produced implementations of MIL-STD-1750.
- Make cost-effective use of existing SEAFAC resources.
- Strive for vendor/implementation independence.
- Recommend a verification approach based on a cost trade-off analysis.
- Provide sufficient descriptive information about the recommended approach so that the Air Force can:
    - Plan future funding and personnel requirements.
    - Write specifications for hardware and software necessary to support approach.
    - Contract for (or develop internally) the necessary facilities.

The Air Force has provided the following guidance regarding the certification objectives:

1. The Air Force desires to maintain compatibility of support software (this implies both assembler source code compatibility as well as object code compatibility in what is equivalent to the problem state).
2. The approach should address a spectrum of computers from the microprocessor to the high performance stand-alone computer.

3. The Air Force desires to maintain an open relationship with vendors with regard to certification plans, programs and procedures.

4. The MIL-STD-1750 Instruction Set Architecture User's Group should be used as the vehicle for effecting changes to the standard.

5. From a practical viewpoint, the certification process for each machine should not exceed two weeks, based on an eight hour day.

6. The Air Force does not desire to define a standard AGE interface as an additional standard. However, it would be acceptable to require scme minimum functional capability, a common I/O interface, as well as a minimum memory configuration to support verification.

7. Only one vendor will be certified at a time.

8. Recertification is not planned. However, retesting may be considered when operational problems are identified or when errors or updates occur in the certification process.

This guidance forms a basic set of assumptions used in this study.

This study investigates verification approaches. IBM has conducted a literature search to identify any novel verification approaches and to gather data from the public domain; the IBM Federal Systems Division team members have visited other IBM locations to gather data about the various approaches inside the corporation and have consulted with the Owego experts concerned with verification.

This document examines the following verification approaches:

- Acceptance Test Program
    - AN/AYK-15A
    - ADAM
    - User Oriented Micro Processor
- Random Instructions
- Analytical Research
- Architectural Verification Program - System 360/370
- Diagnostic

- Functional Test Program
- Instruction Set Processor Specification
- Lockstep

The study methodology uses the following 3-step approach:

Step 1 Apply pass/fail criteria (i.e., feasibility) to each approach.

Step 2 Measure each verification approach which survives the pass/fail criteria for SEAFAC impact, time to perform a verification test, non-recurring start-up costs, recurring costs, and quality of the approach.

Step 3 Select the best approach under which SEAFAC may implement the certification capability.

The recommendation of this study is a certification process which invokes two verification phases to provide a superior means of implementing the certification facility. The first phase uses a deterministic verification program based upon modifications to the AN/AYK-15A ATP. These modifications delete non-MIL-STD-1750 features and add additional tests where required. It is followed by the second phase in which randomly generated sequences of instructions are run on a MIL-STD-1750 simulator and the machine under test: the results are then compared to determine their validity.

The time estimate required to implement this certification process at SEAFAC is a time period of a year and a half; the total cost estimate for the certification process is $1,239,000 using 1979 dollars, $70,000 man year rate, and other Air Force guidelines.

THIS PAGE INTENTIONALLY LEFT BLANK

2.0 SCOPE OF STUDY

The objective of this study is to determine the methodologies and identify the resources required to provide the Air Force with the capability of verifying compliance and certifying vendor produced computers to MIL-STD-1750. This study identifies plans, procedures, hardware support items, support software items, interfaces, personnel, and other resources required to provide ASD/ENASD Systems Engineering Avionics Facility (SEAFAC) with the capability to determine computer instruction set compliance to MIL-STD-1750.

This study addresses various potential approaches to accomplish the certification process, and the identification of hardware, software, interface, personnel, and miscellaneous resources required to support each approach. Each approach was evaluated considering the present capabilities of the existing SEAFAC facility. The advantages and disadvantages of each approach, and their cost impacts were also considered.

2.1 STUDY ACTIVITIES

The following summarizes the program tasks undertaken during this study:

Task 1.0 Identify Approaches: Architecture verification approaches were researched. Approaches used from inside IBM FSD and IBM commercial were surveyed; the Acceptance Test Plan for the AN/AYK-15A was included by direction of the Air Force; and a survey of the literature describing activities in the computer industry was completed.

Task 2.0 Analyze Air Force (SEAFAC) Resources: This task was undertaken to understand the existing SEAFAC resources and identify any newly required resources based on the verification approaches.

Task 3.0 Trade-Off Of Approaches: This task was broken into principally two sub-tasks. In the first sub-task pass/fail criteria (such as feasibility) was applied to each of the approaches. The approaches which passed were then analyzed regarding their Non-Recurring Start-Up costs, and Recurring costs in order to select the best verification approach or combinations of verification approaches.

Task 4.0 Develop Certification Plan: A detailed plan for the recommended approach was developed in this task in order to be used as a guide for implementing the verification capability. The plan includes detailed facility requirements and configurations, a detailed mechanization plan describing how vendors will be able to attain certification, suggested organizational structures for effective control of the certification, detailed designs and procedures to test each function, and impacts to MIL-STD-1750 which pertained to its testability.

Task 5.0 Produce Study Deliverables: This task results in a draft final report, a final briefing, and a final report at completion of the study in accordance with the Statement of Work.

2.2 PERSPECTIVE

The process of computer architecture verification, which is extremely complex for a single manufacturer, is further complicated for the Air Force because the Air Force must deal with multiple manufacturers and formally certify that a machine has passed the architecture verification process.

2.2.1 Architecture Verification

The task of architecture verification is to prove that a given implementation of the architecture performs all functions specified, and executes all combinations of those functions with clearly defined results. This implies not only obtaining the desired results but also ensuring that no other conditions except those specified are performed.

For example, MIL-STD-1750 states for single precision compare:

> "The single precision derived operand, DC, is compared to the contents of RA. Then, the condition status, CS, is set based on whether the contents of RA is less than, equal to, or greater than the DO. The contents of RA are unchanged."

It is important to realize that statements like the above also imply that a number of unstated conditions (e.g., other status bits, registers, etc.) are unchanged. Verifying that unrelated conditions are not affected is as important as verifying expected results.

Items to be checked in each verification test include:

The function
Data patterns
All registers, both used and unused
All interrupts
Main Storage locations, both used and related
Condition codes set or unaffected
Error indicators

Order of occurrence of predicted events
Interference due to simultaneous I/O and/or interrupts

Each architectural verification method has associated with it a degree of quality, which is a measure of how good that method is at finding inconsistencies between the hardware implementation and the architecture specification. This quality may be viewed from three different perspectives:

1. Completeness

2. Coverage

3. Confidence

2.2.2 Certification

The act of conferring certification has associated with it certain liabilities. It is important that the study consider the procedural issues of certification so that these liabilities are minimized. Some examples of these procedural issues follow.

While the ideal goal of certification is to ensure 100 percent compliance with the specification, experience and common sense suggest that.this cannot be obtained at a reasonable cost, (as is discussed later). As a result, less than perfect verification must be accepted.

It is likely that the initial certification capability which is put in place will be subject to improvements over time as experience is gained. Because of a lack of maturity, the verification process as first implemented may falsely indicate compliance and allow a deviation to escape detection. Suppose that Machine A, which has been granted certification, is later retested and found to be non-representative of the architecture. It is recommended that a process be put in place to document this variance and publish that information to any potential user.

The architecture specification itself is not fixed for all time. Rather, changes, corrections, and extensions will be made over time. Again it is recommended that retesting with publication of the results be required of all previously certified computers.

2.2.3 Architecture Specification

It is essential to have a complete, detailed, unambiguous documentation of the architecture to ensure that a compatible verification is possible for all implementations. This document must specify the functions available for use in programming the computer including all significant side affects. It does not tell the hardware designer what techniques or technologies to use in his/her implementation of the architecture. It may include Implementation Notes which describe the intentions behind certain operations or functions and which may aid the designer in any hardware trade-off decisions.

Typically this document includes:

Processor Structure

Main Storage
Addressable Registers
Instructions and Data Format
Addressing Modes
Expanded Addressing Technique
Machine Status
Interrupt System and Status Switching

Instruction Repertoire

Input/Output

Channels
Timers
Discretes
I/O Interrupts
I/O Instruction Repertoire

Protection Features

Multiprocessing Facilities

Multiprogramming Facilities

Security Features

Alternative Subset Implementation

Growth Provisions

2.2.4 Further Refinements to MIL-STD-1750

The Statement of Work in the Air Force's RFP indicates that the study should be concerned with areas of MIL-STD-1750 which require further definition because of testability impacts. Specifically, "The contractor ... shall identify impacts on MIL-STD-1750 which pertain to its testability."

An example of this type of architectural impact due to testability considerations arises from the Lockstep testing approach, which will be discussed in detail in a later section. Basically, this method requires that the computer have a bit in its Program Status Word (PSW) which causes it to enter a trace mode. In the trace mode, an instruction is executed and then the trace interrupt occurs. This trace interrupt saves all the status of the computer in a status area and loads a new PSW. Having a copy of the complete status of the computer at the completion of each instruction execution permits that status to be compared with similar status from a certified computer (that is, one that is certified as a member of the family). Thus, if the LockStep testing approach is chosen, it would impact the definition of MIL-STD-1750 by requiring the definition of a trace mode.

Specific refinements to MIL-STD-1750 will be covered in the recommendation section.

2.2.5 Industry Activities

Computer development has multiple stages in which computer verification is required:

1. As a bring-up tool by Engineers.
2. As a verification tool by Quality Assurance.
3. As a final box checkout by Manufacturing.
4. As a diagnostic check by Field Engineering.

Each area inside IBM was surveyed for their approach to verification and how this could be applied to the Air Force's problem.

2.3 AIR FORCE RESPONSIBILITIES

The Air Force must provide complete, detailed, unambiguous documentation of the architecture (MIL-STD-1750). This is required because informal communication channels are difficult, unreliable, and unmanageable within one manufacturer, and almost impossible across a variety of manufacturers. For example, a "LOAD BYTE" instruction could indicate that a 8-bit byte operand from main storage is loaded into a 16-bit general register; but may neglect to specify what happens to the other 8-bits in that general register. The consequence is that one implementation might clear the 16-bit general register before loading the byte while another might insert the byte. Since the verification program cannot test outside of the specification, the two implementations pass certification and a programmer moving from one implementation to another may have a very subtle error to uncover due to this difference.

The Air Force should extensively test for compatibility of a vendor's implementation to the MIL-STD-1750 specification. It is better to err on the side of conservatism than to under test an implementation. Additionally, the Air Force must be careful not to test outside the MIL-STD-1750 specification as indicated previously.

The Air Force should encourage vendors to have their implementations certified. In order to do this, the certification process should be as painless as possible. For example, the Air Force should freely make available copies of the verification programs for the vendors to "pre-test" their implementations. Furthermore, when the certification process detects a problem, the Air Force should provide as much information as possible about that failure, and then should resume testing for other errors in order to provide the vendor with as much information as feasible about their computer. (However, it must be noted that the information provided need not be to the extent that the Air Force is debugging the vendor machine.)

The Air Force must maintain full documentation of the certification process in order to duplicate the certification process at a later point in time or to reverify that a certification was properly undertaken. When retesting is indicated (as described in another section), the Air Force must document and publish the results.

2.4 STUDY GROUND RULES

The results of this study are based upon the set of ground rules (developed with Air Force guidance) which follow:

1. In SEAFAC, all hardware facilities (in particular the VAX 11/780 computer, the PDP 11/55 computer, and the MIL-STD-1553 interface to these computers) should be considered zero cost from both a Non-Recurring Start-Up (NRSU) and a Recurring basis.

2. The certification process should take less than two weeks. A manned work week is comprised of five 8-hour days; however, an automated process without manual intervention could be available for second and third shifts.

3. Only one vendor at a time would be certified in the SEAFAC facility due to problems associated with proprietary hardware/information about vendor's competitors.

4. It is better to be conservative and over test than to under test during the certification process.

5. In order to be certified, a minimum storage configuration (like 64K) and a standard I/O channel (like MIL-STD-1553 or RS-232) could be required.

6. A standard Ground Support Equipment (GSE) interface is not required because a standard Ground Support Equipment interface for a family of computers starting with a one page micro processor (with RAM) embedded in a system up to a standalone computer in its own box could unfairly penalize the former class.

7. For converting man years into cost, a 1979 labor rate of $70,000 per professional Air Force employee should be used which is also approximately the same for a professional industry employee.

8. During the ten year expected life for the MIL-STD-1750 architecture,

   a. from 313.2K to 522K source lines of assembly code are expected for operational programs, and

   b. twenty computers are expected to be submitted for certification by SEAFAC and ten of these would be resubmitted after initially failing.

9. Programmer productivity is 75 lines of assembler source code per man month and 110 lines of Higher Order Language (HOL) source statement per man month for operational programs. Parenthetically, a 1 to 3 factor is applied for expanding from one HOL statement into assembler code statements. Software maintenance costs are based on:

   a. the existence of two errors per one thousand assembler source lines of delivered code, and

   b. each error requires one man week to repair.

THIS PAGE INTENTIONALLY LEFT BLANK

3.0 DEFINITIONS

This section provides definitions for a number of concepts which are used throughout this report. Definitions are provided here because the terms they define are used quite loosely in the industry. In addition to the definitions, comments are provided which relate the terms to this study.

3.1 COMPLETENESS

Completeness is a measure of how thorough an architectural verification program is in testing that a particular machine meets an architecture specification. A complete verification of an architecture would require checking all possible combinations of memory locations using all possible combinations of instructions and all possible combinations of data patterns, as well as checking all conditions which are to remain undisturbed. The measure could be expressed by the number of combinations tested as a percentage of the number of combinations possible. This is not a very practical measure to apply to architectural verification programs, however, since a complete verification of an architecture such as MIL-STD-1750 would take an unreasonable amount of time. For example, just to check all possible pairs of instructions with each memory location and each data pattern would require ten years (assuming a two microsecond instruction time).

$$(195^2 \text{ pairs} \times 2^{16} \text{ memory locations} \times 2^{16} \text{ data patterns} \times 2 \times 10^{-6} \text{ seconds} = 326{,}632,\ 262.9 \text{ seconds}) > 10 \text{ years.}$$

Obviously, exhaustive testing for a complete verification is not practical.

For most verification tests, the completeness percentage would be a figure near zero. Also, verification Method A could have one hundred times the completeness of verification Method B, while A only exercised half of the instructions and B exercised all of them. For these reasons, completeness would not be a useful measure for distinguishing between architectural verification programs.

3.2 COVERAGE

Coverage is the percentage of the computer hardware (in terms of gates, wires, microcode, memory locations, etc.) which is tested by an architectural verification program. This term is commonly applied to the microcode portion of the hardware. A measurement of coverage has been made for several architectural verification programs. See, for example, Criteria for Architecture Verification, by R. C. Varney and N. P. Groundwater. Coverage for developing a computer (as applied to microcode) is a useful concept, since less than 100 percent coverage implies that parts of the machine remain untested (which may contain errors). When applied to architectural verification, however, coverage has two limitations. First, measurement of the coverage of gates and wires is very difficult. This is significant since failures of gates and wires contribute to architectural discrepancies as surely as do microcode faults. Second, and more important, attainment of 100 percent coverage does not guarantee that no architectural discrepancies remain in the machine. Simply exercising each location of microcode does not mean that every execution path has been taken. Furthermore, coverage is not expected to be a good measure for distinguishing between architectural verification methods, since most methods are easily capable of achieving 100 percent or nearly 100 percent coverage in practice.

3.3 CONFIDENCE

Confidence is the degree of certainty that a given software module (which is known to be correct) will execute properly on a machine which has been certified through the use of an architectural verification program. While confidence would be a useful measure for distinguishing between architectural verification methods, there is no simple means of obtaining estimates of the confidence associated with each architectural verification method.

3.4 QUALITY

Quality is defined as the degree to which an architectural verification method is capable of determining compliance of a hardware implementation to an architectural specification. It is measured by the projected total number of architectural discrepancies expected to remain in a machine after verification. This concept is applied in this study and is discussed in detail in Section 5.

3.5 SECOND-ORDER EFFECTS

Second-order effects are program execution errors which are the result of the interactions between two instructions. For example, the ADD and MULTIPLY instruction could function properly when executed independently, but fail when executed sequentially. Second-order effects are both very real and very hard to find. The definition of a completeness suggests the difficulty in testing all instruction pairs. Obviously, even higher-order effects are possible.

3.6 ACCEPTANCE TEST PROGRAM

The Acceptance Test Program (ATP) is the software portion of the Acceptance Test Procedure which is used to sell-off the hardware. This program attempts to verify that all functions defined in the hardware specification, which can be observed by the programmer, perform properly. An ATP usually contains verification tests, performance tests and sections used for detailed measurements.

3.7 ARCHITECTURAL VERIFICATION PROGRAM

An Architectural Verification Program (AVP) is a computer program used to demonstrate that a specific hardware implementation of a computer architecture performs all programmer observable functions as defined in the architecture. All basic architectural features are assumed to function properly and are used freely throughout testing. An AVP usually not only verifies that what is expected is performed, but also that no unintended functions are performed.

3.8 DIAGNOSTIC PROGRAM

A Diagnostic Program is a computer program written to verify the proper operation of the hardware and to attempt to isolate any failures (typically, the Shop Replaceable Unit level). The size of the program is reduced by knowledge of the hardware implementation and by the addition of hardware and/or microcode to assist it in achieving its objectives. The program is highly implementation dependent.

3.9 FUNCTIONAL TEST PROGRAM

A Functional Test Program (FTP) is a computer program which is used for initial debug of hardware. It tests all functions of the hardware observable to the programmer. An FTP is written under the premise that all architectural features require testing prior to use. It may take advantage of the hardware implementation to reduce the size of the program.

3.10 COMPUTER HARDWARE DESCRIPTION LANGUAGES (CHDL)

CHDLs provide for precise syntactical descriptions of the functional behavior of digital circuits. They are used to describe the logic gate networks, sequential circuits, modules, their connections, and their control in a digital system.

Examples: Instruction Set Processor (ISP);
  Used in ISP Verification Approach
Language for Symbolic Simulation (LSS);
  Used in Analytic Verification Approach

3.11 CERTIFICATION PROCESS VERSUS VERIFICATION PROGRAM

Certification is the process of testing a computer by using a verification program in order to determine that the computer conforms (or fails) to an architectural specification like MIL-STD-1750.

4.0 STUDY METHODOLOGY

This section provides a discussion of the reasons behind the selection of the cost model approach and provides the rationale for the elimination of other approaches. It describes the criteria that are used to evaluate the various architecture verification methods and presents a systematic approach that is used to trade-off those methods. This section also describes the methodolcgy that was applied in this study for data gathering and data analysis.

4.1 GOALS OF THIS STUDY

In IBM's proposal to the Air Force, a number of verification methods were discussed. These approaches were:

1. Acceptance Test Procedures for Existing MIL-STD-1750 Related programs (AN/AYK-15A, ADAM, User Oriented Micro Processor)
2. Random Instructions
3. Analytical Research
4. Architectural Verification Program - System 360/370
5. Diagnostic
6. Functional Test Program
7. ISP
8. Lockstep

The purpose of this study is to determine the advantages and disadvantages of each of these verification methods (in light of SEAFAC resources), to systematically evaluate these advantages and disadvantages to select the best alternative, and finally, to identify the detailed designs and procedures needed to equip SEAFAC with the capability of determining whether or not any machine meets the requirements of MIL-STD-1750. Sufficient descriptive information is provided on the implementation of the recommended approach so that the Air Force can: plan future funding and personnel requirements, write specifications for the hardware and software necessary to support the approach, and contract for (or develop internally) the necessary facilities. A summary of the study approach is shown graphically in Figure 4-1. The verification approaches are evaluated against the following criteria:

1. Impact to SEAFAC Resources - necessary additions, such as special test equipment or specially trained personnel must be viewed as a cost.

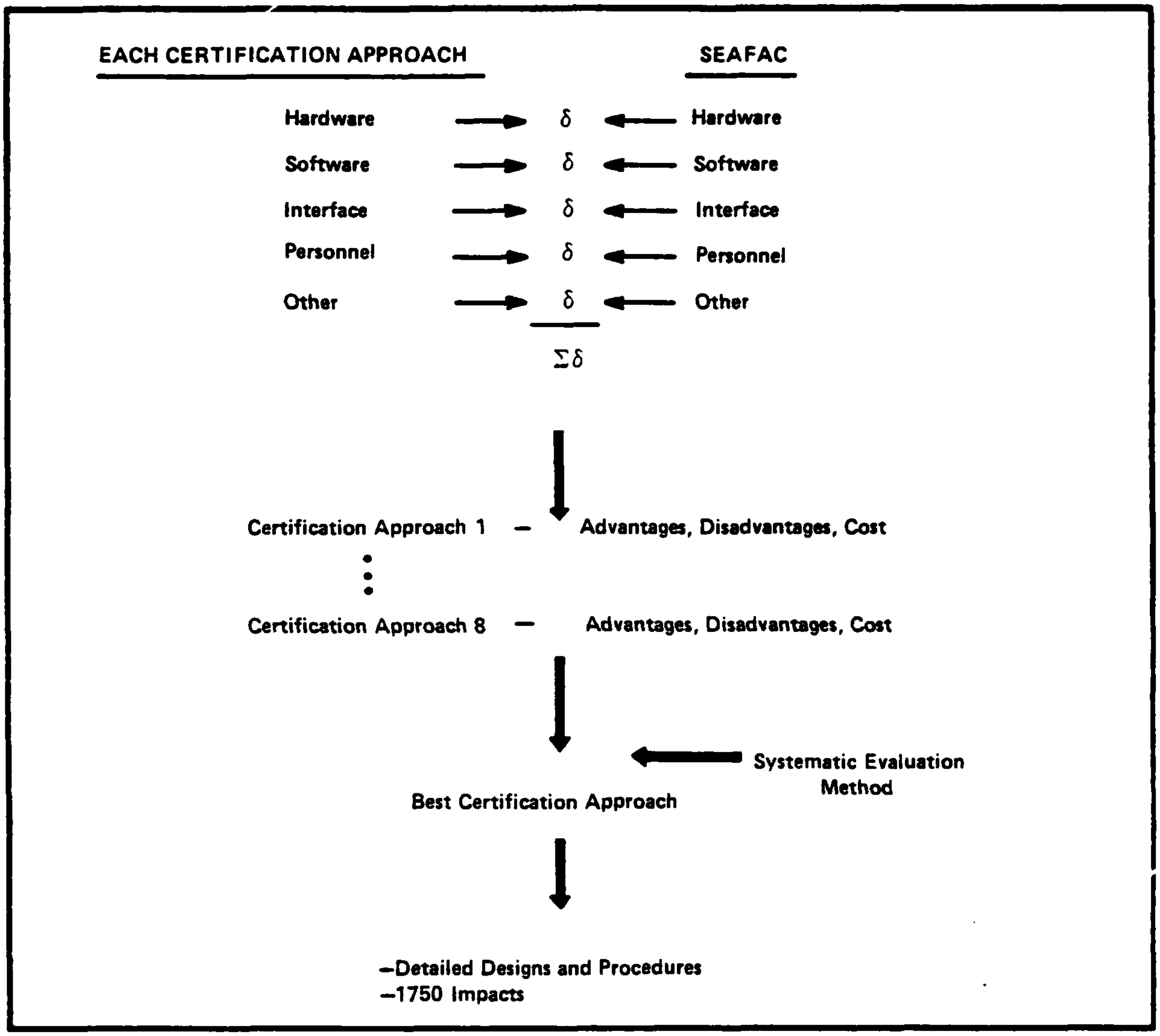

EACH CERTIFICATION APPROACH
SEAFAC
Hardware
Software
Interface
Personnel
Other
δ
Σδ
Certification Approach 1
Advantages, Disadvantages, Cost
Certification Approach 8
Advantages, Disadvantages, Cost
Systematic Evaluation Method
Best Certification Approach
–Detailed Designs and Procedures
–1750 Impacts

Figure 4-1. Summary of Study Approach
(Sheet 1 of 1)

2. Time Required to Perform Validation Effort - especially lengthy procedures invoke labor costs and monopolize valuable test resources.

3. Non-Recurring Start-Up (NRSU) Costs - includes items such as writing architectural verification software and test procedures.

4. Recurring Costs - includes items such as architectural verification software maintenance, and staffing during the certification process. This figure is develcped as the cost per certification.

5. Vendor Independence - the verification method under consideration must not discriminate on the basis of vendor-unique items which are not specified by the MIL-STD-1750 architecture. For example, a method which requires that a particular type Ground Support Equipment interface be present to perform the test sacrifices vendor independence.

6. Application Independence - the verification method under consideration must not discriminate on the basis of items which are dictated by the application requirements, but which are not specified by the architecture, e.g., machine weight, power, volume, or cooling requirements.

7. Quality - the degree to which the method verifies that the hardware actually reflects the architecture. The quality criteria encompasses the notions of Completeness, Coverage, and Confidence defined earlier. These concepts are discussed in detail in Section 5.

8. Cost of Software Revalidation - if a validation method is of less than perfect quality, there is some finite probability that any software developed in consonance with MIL-STD-1750, even software previously validated on a certified machine, can fail on a second certified machine used for a new application. It would therefore be necessary to revalidate any code which will be used on this second machine.

After evaluating the different verification methods against these criteria (the methodology of which will be discussed later), the next step is to systematically analyze the results and select the best method for SEAFAC.

4.2 THE TRADE-OFF APPROACH

The following evaluation method is employed in this study. First, the VENDOR INDEPENDENCE and APPLICATION INDEPENDENCE criteria are treated in a pass/fail manner. Because it is essential that the verification method employed test any implementation of MIL-STD-1750, any method which fails these criteria is rejected and is not subjected to further consideration.

The remaining criteria, SEAFAC IMPACT, TIME, NRSU, RECURRING ELEMENTS, QUALITY, and SOFTWARE VALIDATION ELEMENTS, are converted to costs. (How this is accomplished is discussed in Section 5.) Each verification method that survives the pass/fail criteria is measured against these criteria. Since each criteria represents a cost, a total cost is obtained for each method. The method with the lowest cost is selected as the best.

While some of these criteria are easily recognized to be of a cost nature, others do not have an obvious cost relationship. Therefore, each of these criteria, and the techniques that will be used to measure them, will be discussed in detail.

4.3 DATA COLLECTION

The first step in the study is to collect data on all of the verification approaches. This is necessary to fully understand each of the approaches and to support the cost model.

Data collection was accomplished in five phases: literature search, interviews and site visits, ECs and Validation costs, local cost estimation, and miscellaneous trips.

Use of the cost model required that the following types of data be gathered for each of the verification approaches:

Pass/Fail Data
- vendor independence
- application independence
- feasibility
- uniqueness from other approaches
- availability of information
- testable within a two week period

Cost Data
- non-recurring start-up costs
- recurring costs
- SEAFAC resources

Quality Data
- Engineering Changes (ECs)
- Software validation costs

(The reasons for collecting the quality data are discussed in Section 5.) In addition, detailed descriptive information about each of the verification approaches is required.

4.3.1 Literature Search

An intense literature search has been made to support the research being done on the MIL-STD-1750 Certification Study. This research provided additional data regarding the proposed approaches; but did not uncover any new verification approaches to be investigated. The following literature data bases were queried by IBM Technical Information Retrieval Center:

NTIS	-	National Technical Information Service
INSPEC	-	Information Service in Physics Electrotechnology and Control
IBM	-	IBM Internal Technical Reports/Research Reports
EIXF	-	Engineering Index

The results of this search can be found in the Bibliography. This lists the papers and articles directly relevant to this study.

4.3.2 Interviews and Site Visits

All of the verification approaches investigated (except MIL-STD-1750 ATPs) have been used within the various IBM Divisions. To obtain the detailed information required, the Divisions shown in Figure 4-2 were consulted. Users of each of the approaches were interviewed and, where possible, the originators of the concepts were interviewed. (This was the case for the Lockstep, Random, and Analytical approaches.) Site visits were made to the T. J. Watson Research Center at Yorktown, N.Y., the System Products Division at Poughkeepsie, N.Y., the System Products Division at Endicott, N.Y., and the General Systems Division at Boca Raton, FLA.

4.3.3 Engineering Change and Software Validation Cost Data Collection

Engineering Change data were gathered with help of the Owego Part Number Identification User System (CPIUS). This automated system permits a user, via a display terminal, to obtain Engineering Change and part number information for the avionics computers built by IBM in Owego, N.Y. Engineering Changes of interest were physically pulled

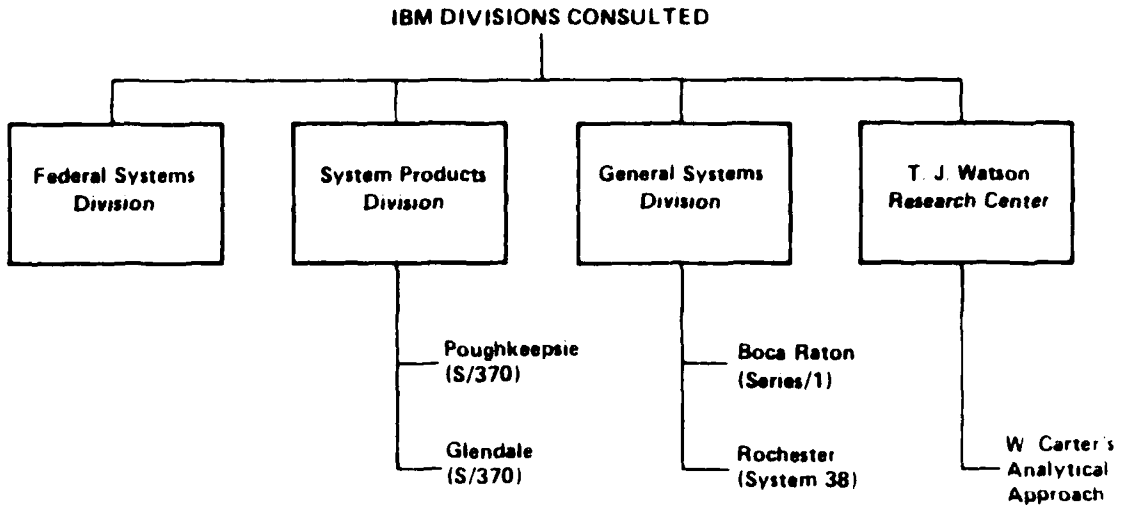


Figure 4-2. IBM Divisions Consulted
(Sheet 1 of 1)

from storage and scrutinized to determine their applicability to the study.

Software validation cost information was obtained by interviews with the software personnel responsible for the validation efforts. These personnel were located in the Air Force, the Navy, and IBM.

4.3.4 Internal Data Gathering

To supplement the cost data gathered through the fact finding trips and extensive literature search, various IBM - Owego sources were utilized. FSD personnel experienced in support software development, systems integration, and hardware testing and verification contributed cost data and other information in their area of expertise. This data is reflected in developing the cost figures related to the various approaches described in Section 7.

4.3.5 Miscellaneous Trips

Two other trips were taken by team members in support of this study. The first was to SEAPAC to obtain information about the equipment, personnel, and facilities available to support the MIL-STD-1750 certification effort. The second trip was to Palo Alto, California to attend the 1979 International Symposium on Computer Hardware Description Languages and their applications. This allowed the latest available data on the CHDL's approach to be included in this study.

4.4 DATA ANALYSIS

Analysis of the gathered data began with the application of the pass/fail criteria. Additional criteria to those discussed in the proposal were identified. (The cost model was then applied to those verification approaches which survived the pass/fail test.)

The Engineering Change data and software validation cost data were analyzed to determine the cost penalties which will be incurred because of the necessity of using a non-perfect certification method. This information, combined with the cost data above, form the basis for the study recommendation.

5.0 COST MODEL

To summarize the analysis described in Section 4, three things must be done to select the best verification method:

1. Apply the pass/fail criteria,

2. Estimate the costs for the remaining approaches, and

3. Treat in an ad hoc manner any newly identified criteria which do not fit into either (1) or (2).

This section describes, in some detail, the concepts behind the cost estimation process (item (2)). The application of these concepts to the gathered data is discussed in Sections 6 and 7.

This section is broken into two parts. The first part discusses the following items: SEAFAC IMPACT, TIME, NPSU, and RECURRING COST. These items are relatively easily converted to dollar expenditures, which are necessary for SEAFAC to establish, use, and maintain a certification facility for MIL-STD-1750.

The second part of this section discusses QUALITY and SOFTWARE VALIDATION COSTS. These items represent costs which will be incurred over the lifetime of MIL-STD-1750 and which are in the form of a penalty which will be paid by the individual programs which use MIL-STD-1750. They are not direct costs to SEAFAC, but are true costs to the Air Force, nonetheless. Conversion of QUALITY to cost is more involved than the conversion of the other items to cost. As a result, a significant amount of this section is devoted to discussion of the QUALITY concepts.

5.1 COSTS TO SEAFAC

This section describes the evaluation criteria SEAFAC IMPACT, TIME, NPSU, and RECURRING COST. These items will be a direct cost to SEAFAC. The descriptions provided here are in the form of a priori expectations; they are included here as exemplary information for the cost model. Again, the actual findings of the study appear in Sections 6 and 7.

5.1.1 Impact to SEAFAC Resources

SEAFAC contains a great deal of resources, including both equipment and personnel, which could potentially be useful for architecture verification. However, a particular verification method might require, for example, some unique test equipment to interface with the unit under test. A cost would be incurred for procuring that equipment and training personnel to use it.

Another verification method might require that SEAFAC hire an architecture expert. A cost would be incurred to hire this person, pay his salary, and pay his benefits.

Each verification method is examined for any such unique considerations.

5.1.2 Time Required to Perform Validation

The time required to perform the certification/verification approach translates into a cost for personnel and equipment (required to support the rate of testing and maintenance).

5.1.3 Non-Recurring Start-Up Costs

Many types of one time start-up costs might be incurred with the various verification test methods. It is possible that a particular method would require that some form of architecture verification program either be written or modified. Detailed test procedures would likely have to be written for any approach. For the Lockstep approach, it may be necessary to procure a known, standard machine. The ISP generated AVP method may require development of AVP generation software. The Diagnostic approach might require analysis of each manufacturer's programs. Another requirement may be some type of Ground Support Equipment (GSE) or I/O interface to connect a unit under test to the test hardware.

5.1.4 Recurring Costs

Software that is to be written must also be maintained. The same applies to any hardware to be procured. These costs are, of course, related to their useful lifetime. Recurring costs are also incurred due to the staffing during the validation process.

5.2 QUALITY AND VALIDATION COSTS

It is important to recognize the significance of the quality measure of any architecture verification method because no method of testing a machine will be perfect, and, as a result, costs will be incurred.

Quality is defined as the degree to which an architectural verification method is capable of determining compliance of a hardware implementation to an architectural specification. The unit of measure is the expected number of architectural discrepancies remaining in a machine after architecture verification. The quality of a given verification method, then is an assessment of how good that method is at finding machine defects or architectural discrepancies.

The concept of quality is related to three separate concepts which are widely used in the industry. These are completeness, confidence and coverage. Completeness requires checking the machine for all possible data types with all possible instruction sequences in all possible memory locations, etc. Hardware coverage is the percentage of a machine that is tested in terms of gates, memory locations, microcode, wires, etc. Confidence is the degree of certainty that a software module will execute properly on a verified machine. These concepts have been briefly described here because they are commonly referred to in industry. None of these three concepts are used in the cost model, however, for reasons of practicality (discussed in Section 3). Rather, the concept of quality will be used.

One would expect that the direct cost of implementing a verification method would be roughly related to the quality of that method as shown in Figure 5-1. A test that was not very extensive (and, hence, not of high quality) would not be difficult to program and would not take long to run, making it low in cost.

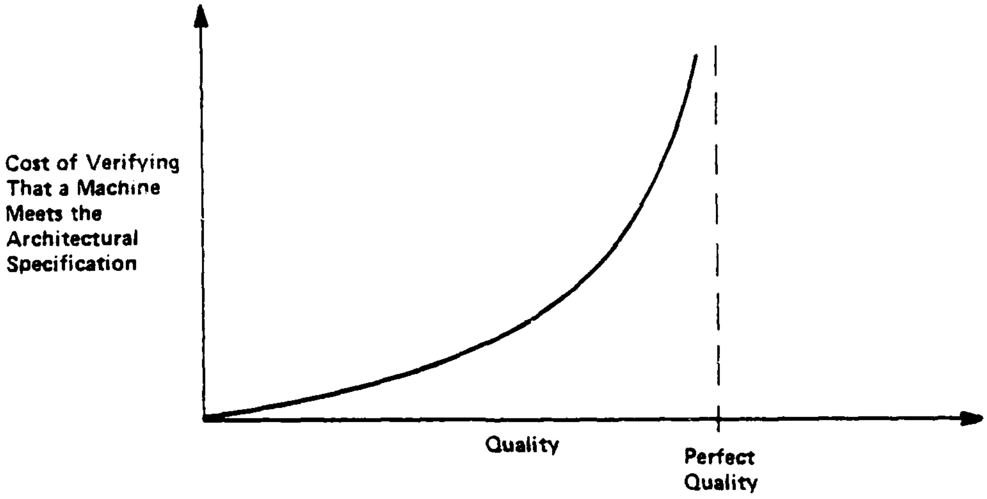


Figure 5-1. Verification Cost as a Function of Quality of Test

Slight additional investment to this minimal test would result in large gains in quality as more and more instruction types and formats were tested. The cost of approaching perfect quality, however, would be extremely large, because it would be necessary to test all possible data combinations with all possible instruction sequence combinations in all possible memory locations, etc.

5.2.1 Reasons for Considering Quality

Achievement of perfect quality in a verification method would require that method performs exhaustive testing. That is not feasible for MIL-STD-1750 certification (see the definition of Completeness).

Since any useable verification test method is of less than perfect quality, there is a very real possibility that the tested machine will still have undiscovered architectural discrepancies, and that any operational software will not execute correctly (that is, as defined in the architectural specification manual). It is necessary, therefore, to check all software that is to run on that machine for proper execution, i.e., to validate that software. Typically, this validation is initiated in a laboratory simulation and then completed in actual flight test.

The software to be validated can be broken up into two classes: existing software that is to be captured for this machine (e.g., navigation modules, executives), and new software that is to be written for this machine.

The cost associated with revalidating the existing software (which had already been validated for some other machine) is a direct result of the less than perfect quality of the verification method. If a verification method were 100 percent complete in certifying that a machine met MIL-STD-1750, any operational software would execute properly and, hence, revalidation would be unnecessary. However, since no verification method is of perfect quality, revalidation is necessary. Any architectural discrepancies which are discovered during operational software revalidation (which remained undetected at the time of verification) require corrections. These corrections extend the revalidation process and increase the cost of validation. The quality of the verification approach will determine the number of architectural discrepancies which remain undetected at the time of architectural verification, which in turn, determines the number of architectural discrepancies which will be discovered during software validation. Therefore, the cost of revalidation is a function of the quality of the test which certifies the machine, since the direct result of an imperfect verification is the oversight of architectural discrepancies which must be fixed and which extend the revalidation process. (Presumably, software errors have been discovered in the initial validation effort.)

It is expected that these costs could be substantial. If an architectural discrepancy is discovered in the flight testing phase of software validation, a significant recovery process is required. First, sufficient data must be collected to permit identification of the hardware fault. The computer must be physically pulled from the airframe. An engineering fix for the problem must be devised. This fix must then be installed in the machine. The machine must then be retested and installed in the airframe. Additional flight tests must be performed to verify that the engineering change both solved the problem at hand and did not create any new problems. The software which had been validated to date must be re-checked for proper performance. All of these activities can add up to a substantial penalty in time and money.

The other class of software that needs to be validated is new software to be written. Since new software must be validated in any case, one might argue that this is an irrelevant consideration. However, the cost of the validation will certainly be dependent on the quality of the test method used to certify the machine. A machine that has been poorly certified may have many architectural discrepancies which must be found and corrected in the course of operational software validation and, conversely, a well certified machine will greatly reduce the exposure to unanticipated software validation costs because few architectural discrepancies will remain.

New software validation costs may be divided between costs incurred in correcting architectural discrepancies and costs incurred in correcting software errors (shown in the right-hand side of Figure 5-2). The new software validation costs of interest here, i.e., those associated with the quality of the method used to certify the machine, are only the costs of finding architectural discrepancies (software errors have nothing to do with the quality of the hardware verification). Old software revalidation costs are a good approximation for these new software costs since they are a measure of the costs associated with finding architectural discrepancies only (see the left-hand side of Figure 5-2).

	Existing Software (to be captured)	New Software (to be written)
Software Errors	None. Software previously validated	No. Costs exist, but are of no relevance (not related to quality)
Architectural Discrepancies (Hardware)	Yes	Yes

Figure 5-2. Quality-Related Cost Components of Software Validation

In summary, the implication of less than perfect quality of a verification method is that additional software validation costs will be incurred. The magnitude of these costs is a function of the degree of quality.

Also note that it is only necessary to revalidate operational software, not support software. A Program Manager only cares that the software which runs on his machine is correct. A error in support software which affects the operational software would be caught during revalidation. Validating the operational software will, in effect, validate the support software, at least as far as the Program Manager is concerned. Errors in support software which do not affect the operational software are irrelevant.

5.2.1.1 Quality Measurement

In order to project software validation costs, two items must be estimated - the relationship between quality and the cost of software validation, and the degree of quality associated with each verification method. While these are not easy tasks, ignoring quality and software validation costs (which are real costs) will bias the study results in favor of the cheaper, less complete verification methods and would result in ignoring substantial costs to the Air Force. The following method is used to estimate quality.

The measure of the quality of a verification method is defined as the number of architectural discrepancies that remain in a machine after that machine has been certified as being representative of the architecture. That is, how good was the test at catching all of the architectural discrepancies in the machine? (Implicit in this measure is the notion that, for the purpose of architecture verification, all machine defects, regardless of their cause, are failures to meet the architecture. A burned-out gate which causes a multiply failure on a particular data combination is as much a failure to meet the architecture as is a defective algorithm which causes a multiply failure on a particular data combination.)

If a new machine is designed and built to a particular architecture, and then tested with some architectural verification method, many architectural discrepancies would be discovered initially in the hardware development phase. As time went on, the number of architectural discrepancies discovered would diminish. Finally, no more architectural discrepancies would be discovered and the machine would be deemed architecturally correct, at least as well as the verification method could determine. The machine would then be "sold-off". However, experience would suggest that some hidden architectural discrepancies remain in the machine. Further use of the machine, such as in software validation or in actual flight operation, would uncover some of these hidden architectural discrepancies. The rate of discovery would jump after "sell-off", and again, the rate of discovery would be a decreasing function of time and would gradually approach zero. This error discovery scenario is depicted in Figure 5-3. The number of architectural discrepancies remaining after verification (sell-off) is the cross-hatched area.

For machines already sold-off and in use, it is possible to estimate the number of undiscovered architectural discrepancies that remain at sell-off. The number of architectural discrepancies remaining in a machine after architectural verification is the sum of those architectural discrepancies discovered to date (since sell-off) plus those (as yet) undiscovered architectural discrepancies. The former are recorded in the Engineering Changes (ECs) written against the machine. The latter may be estimated by plotting the number of ECs with time, fitting and extrapolating a curve, and integrating over time from the last data point until infinity. This is shown graphically in Figure 5-4.

The number of architectural discrepancies found will be a function of two factors - complexity of the machine and quality of test. The organizational/architectural complexity is a factor relating the number of remaining errors with the quality of the verification method. One would expect that the number of errors remaining after certification would be larger for a larger machine, independent of the verification method. A more valid quality measure is therefore obtained by dividing the number of architectural discrepancies remaining in the machine by the organizational complexity, which can be approximated by the size of the machine (discussed in Appendix B).

The resulting quality measure, then, of any particular verification method i, as measured against machine j is expressed as

$$\text{quality}_i = f\left[\frac{\text{ECs since sell-off}_j + \text{projected remaining discrepancies}_j}{\text{machine size}_j}\right]$$

Multiple data points for a given method are averaged to generate a composite quality estimate for that method.

5.2.1.2 Engineering Changes and Architectural Discrepancies

For the purposes of this study, the concept of what constitutes an architectural discrepancy is less restrictive than the concept normally used by the industry. Any machine change which, if unprocessed, would cause the machine to fail an architecture verification test is considered to be an architectural discrepancy. This includes things not normally associated with instruction set architecture, such as electrical noise, timing races, and logic errors. Also included, of course, are more obvious

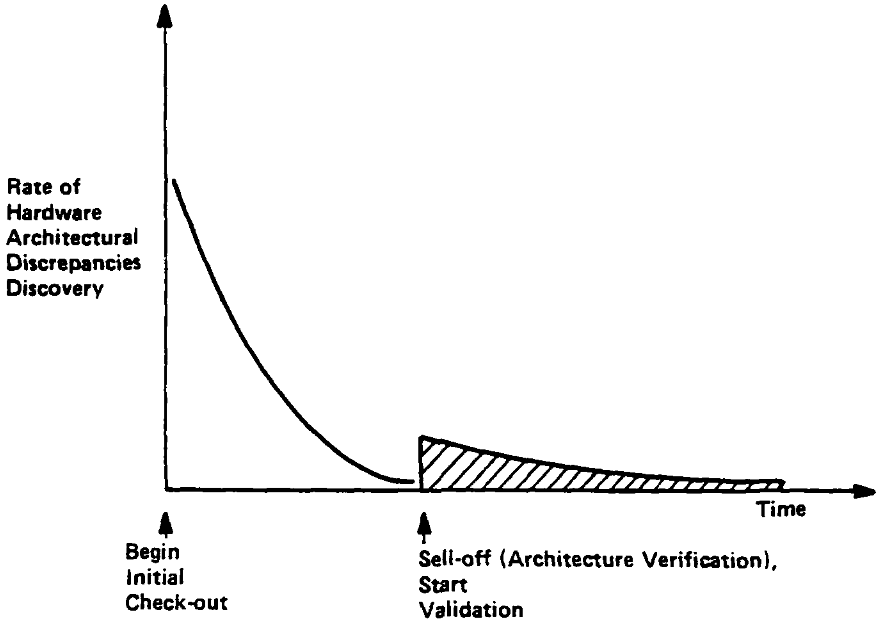


Figure 5-3. Discovery of Architectural Discrepancies Over Machine Lifetime

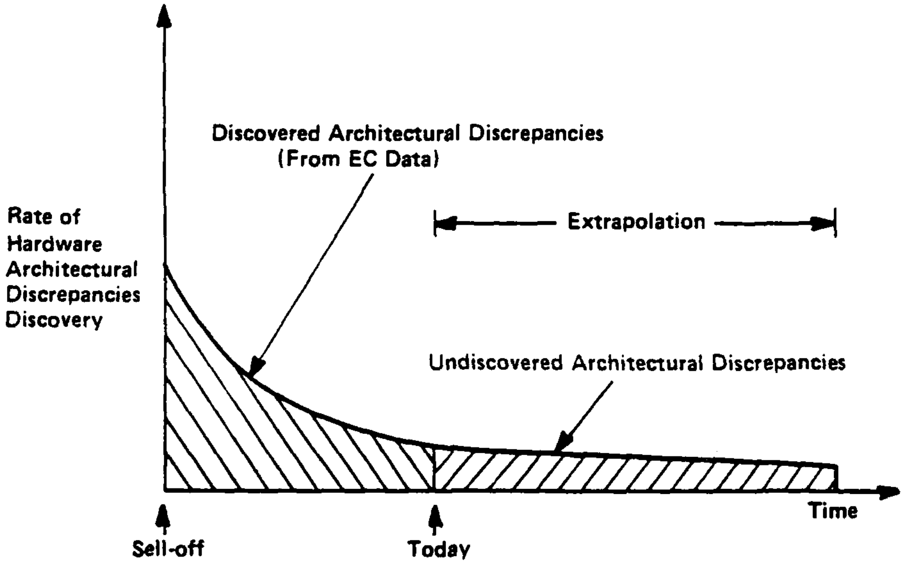


Figure 5-4. Hardware Architectural Discrepancies Remaining After Architecture Verification

architecture-related problems such as defective algorithms or improperly implemented microcode.

The reason for this permissive definition is two-fold. First, consider the certification process to take place at SEAFAC. The Air Force cannot discriminate between types of failures for a machine which does not properly execute a test. Any failure must be a failure to meet the architecture, regardless of the cause. The second reason is that all of these failures (including those not normally associated with instruction set architecture) will be a true cost to the Air Force during Operational Flight Program validation.

This is not to suggest that all Engineering Changes count as architectural discrepancies, however. ECs are written for many reasons including changing screws, coatings and connectors, as well as for customer directed changes and cost reductions. These types of changes are not relevant to this study. As a result, each EC must be scrutinized as to its relevance.

5.2.1.3 Estimating the Quality - Validation Cost Relationship

As discussed earlier, the costs of validating operational software can be broken into two parts - the cost associated with correcting software errors and the cost associated with correcting hardware errors (architectural discrepancies). Only the latter cost is of interest in this study, since the cost associated with correcting software errors is independent of the quality of the verification approach. As used in this study, the term "validation cost" means only those validation costs associated with architectural discrepancies.

The expected relationship between the degree of quality and the cost of validation is shown in Figure 5-5. The rationale for the shape of this curve is as follows. A near-perfect test would incur a basic cost for validation (e.g., flight test), but should proceed smoothly with few, if any, defects to be corrected. Less perfect tests would require more extensive validation efforts, thereby incurring higher costs.

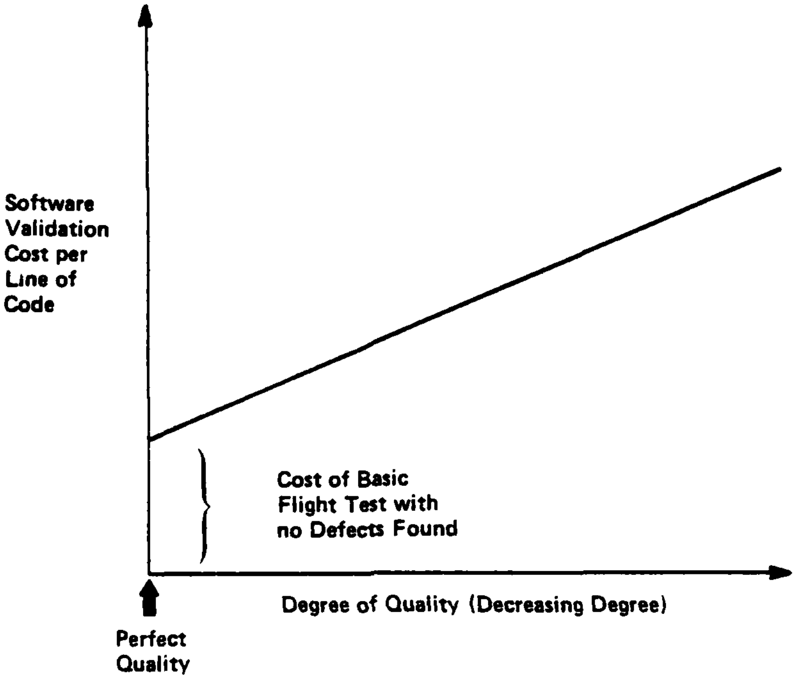


Figure 5-5. Validation Costs as a Function of Quality

The quality - software validation cost relationship can be estimated by collecting both software revalidation cost data and quality data on programs that have incorporated hardware upgrades to existing machines with the intention of capturing existing operational software. This yields a curve which has only those costs which are quality related; the software validation costs associated with correcting software errors are thereby factored out. Since the existing software would have already been validated, the cost of revalidation could be attributed solely to those architectural discrepancies which remained in the machine upon completion of the architectural verification process. That is, the imperfect quality of the verification method was the only reason that revalidation costs were incurred. (Note that the gathered cost data must be scrutinized to see that no costs are included for the addition of functional capability.)

It is not necessary to obtain quality versus software validation cost data for each verification method. A machine that has been verified to a certain degree (i.e., according to the quality of the method used) will incur a software validation cost commensurate with that quality, regardless of the verification method used. An architectural discrepancy has a cost of correction which is independent of the verification method used on that machine. Hence, the quality - software validation cost relationship, regardless of which verification method was used to estimate that relationship, can be applied to all validation methods.

The software validation costs are a function of the amount of operational software to be validated. An increase in the number of lines of code would result in a proportional increase in the cost of validating that code. Therefore, the software validation cost must be normalized by the number of the lines of code.

5.2.1.3.1 Use of the Quality vs Cost Relationship. Application of the measured quality of the different verification approaches to the quality versus cost relationship yields a cost (of having less-than-perfect, quality) for each of the verification approaches.

To compute these costs, EC data are gathered for each of the verification approaches. Projections are then made to estimate the total number of architectural discrepancies expected to remain in the machine after verification. (These two steps are performed in the same manner as is used to estimate the quality versus cost relationship.) These projections must then be normalized by the size of the machine (discussed in a later section). Finally, each of these normalized projections is applied to the quality versus cost relationship to obtain the cost for each of the verification approaches.

5.2.1.3.2 Summary of the Quality Concepts. Quality is defined to be the degree to which an architecture verification method is capable of determining compliance of a hardware implementation to an architectural specification. No verification method is of perfect quality. Less-than-perfect quality implies that architectural discrepancies will remain in a machine after verification testing is successfully completed and certification is granted. These architectural discrepancies will result in a cost to the Air Force in the form of an extended operational flight software validation process.

Estimation of the cost penalty due to less-than-perfect quality for the different verification methods is basically a two-step process:

Step 1. Estimate the cost versus quality relationship by

(a) gathering EC data for various hardware upgrade programs,
(b) gathering software validation costs for those same programs, and
(c) fitting curves to the data of (a) and (b).

Step 2. Gather EC data for the different verification approaches and apply to the cost versus quality relationship obtained in Step 1.

5.2.1.4 Objections to the Quality/Software Validation Cost Approach

It may be argued that the quality data will be a biased measure of the true quality of any given verification method. For example, consider the Architecture Verification Program (AVP) method. AVP development is typically a two step process. The first step is to write the program based on the architecture specification document (Principles of Operation Manual). This will test a certain percentage of the machine (i.e., it will have a certain level of quality). Since it is impossible to test all possible data combinations and since many failures are data related, a second step is then performed and the AVP is "fined-tuned" to the data sensitivities of the particular machine under test. The machine is examined to verify that all possible execution paths (which are data dependent) are exercised (i.e., there is 100 percent microcode coverage). For example, a multiply algorithm would be examined to verify that the different test cases used all possible branch paths in the algorithm.

This fine-tuning affords an increase in quality for most approaches (except Random) which is unavailable to the Air Force since it cannot be determined in advance what algorithms will be used by different manufacturers to implement the instruction set. This results in an upward bias of the quality estimate.

While this is a valid objection, it does not severely limit the value of this data, however, because of two factors. First, it is probably a uniform bias. That is, each program to be investigated has as its goal a high degree of quality. It is reasonable to expect that each program used this relatively inexpensive method (i.e., fine-tuning) for improving test quality. Second, it is the primary goal of this study to select the best method <u>relative</u> to the others.

Another objection to this quality - software validation cost approach is that a large number of data points would be necessary to obtain a high degree of accuracy in cost estimation. This is also a valid objection. However, the alternative is to ignore quality and software validation costs; but experience would suggest that these costs can be substantial. It is better to make use of the information that is available (accepting a degree of uncertainty) than to ignore it.

5.3 PROJECTION OF THE TOTAL NUMBER OF ARCHITECTURAL DISCREPANCIES

Appendix B describes in detail the method for projecting the total number of architectural discrepancies from the EC data. It also describes the method for normalizing the data by machine size.

6.0 VERIFICATION APPROACHES

A number of different approaches have been identified for use in verifying computer architectures and have been investigated:

AN/AYK-15A Acceptance Test Program (ATP)
Random Instructions
Analytical Research
Architectural Verification Program (AVP) - System 360/370
Diagnostic
Functional Test Program (FTP)
Instruction Set Processor (ISP)
Lockstep

Each of these approaches is described herein as it currently is used, along with observations regarding its applicability to MIL-STD-1750. Pass/fail criteria are also applied to each verification approach. A subsequent section describes these approaches after they are modified to apply to MIL-STD-1750. Any special requirements for implementation are also described. Finally, the methods are compared and the essential differences among them are clarified.

6.1 AN/AYK-15A ACCEPTANCE TEST PROGRAM

6.1.1 Description

The AN/AYK-15A Acceptance Test Plan (ATP) consists of a number of tests which must be successfully completed in order to sell-off an AN/AYK-15A computer. These tests include testing the MIL-STD-1750 architecture among other tests. The ATP method consists of extracting the portion of the ATP which tests the MIL-STD-1750 architecture and using this as the MIL-STD-1750 architecture verification program.

The AN/AYK-15A ATP consists of a number of subtests which must be passed as part of the sell-off procedure for the AN/AYK-15A computer. These subtests include: User's Console, Instruction Set, Registers, Main Storage, Bus Controller, Input/Output, Power ON/OFF Sequencing, etc. The ATP includes some subtests which are not part of MIL-STD-1750. These subtests (User's Console, Bus Controller, Power ON/OFF Sequencing) would be eliminated from the ATP to make it a vendor independent architecture verification program.

The Instruction Set test verifies OP CODE assignment and basic functional operation of each instruction. It then checks that the instruction correctly modifies, or does not alter, as the case may be, the status word for all possible values the status word may have. This portion of the test is exhaustive. It also verifies the indexing capability for all instructions which allow indexing. It checks the use of each of the 15 index registers and uses positive, negative and zero values in the index registers. It also verifies the capability of each instruction to generate an underflow and/or overflow interrupt, when required. Data values used in the Instruction Set test cover all combinations of positive and negative numbers.

The test also has a benchmark portion to verify instruction times. This test would be eliminated, since no timing is specified in MIL-STD-1750. There is also a random instruction hang test to verify the handling of illegal instructions.

The Register test verifies the capability to address, set and reset various registers in the AN/AYK-15A. Only those registers defined in MIL-STD-1750 would be included in this method. The registers checked include the CPU general registers, Memory Protect RAM, Status register, and Interrupt Mask register. The CPU general registers and the Status registers are checked by running all 64K possible data patterns through them.

The Main Storage test verifies each main storage location by writing and reading each of the 64K possible data patterns in each location. The test also verifies the main storage protection features.

The Input/Output tests verify the operation of the Timers, Direct Input/Output, external interrupts and DMA.

6.1.1.1 Block Diagram

Figure 6-1 depicts a hardware block diagram for this approach.

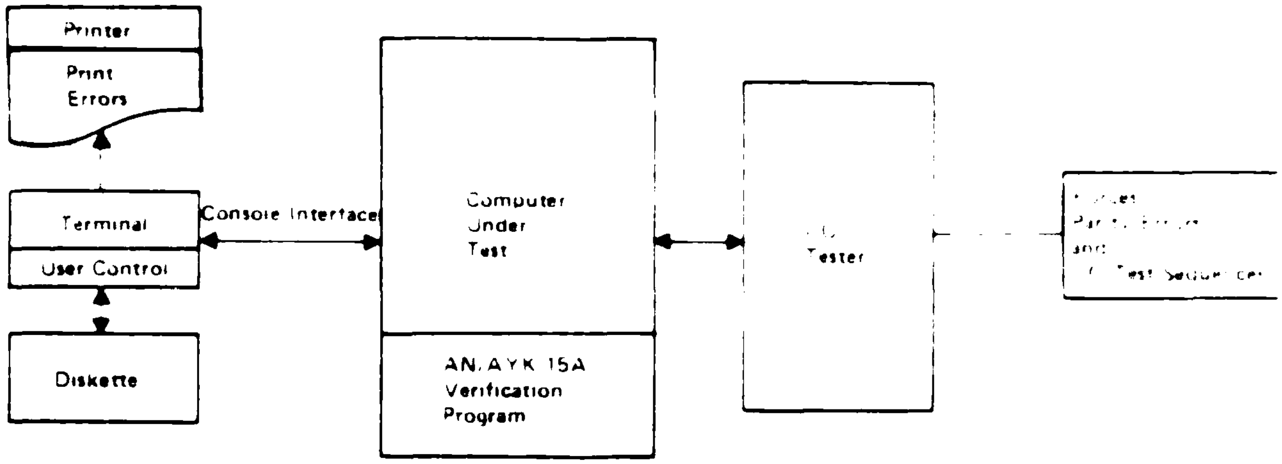


Figure 6-1. AN/AYK-15A ATP Block Diagram

6.1.2 Experience

The ATP approach has been used by many manufacturers, including IBM, to verify their computers.

6.1.3 Hardware Resources

The AN/AYK-15A ATP requires the console (which is optional in the current version of MIL-STD-1750) and special hardware to test the external interrupts and DMA. To work in a multiple vendor environment, the hardware interfaces would have to be standardized.

6.1.4 Software Resources

The software resources for the AN/AYK-15A already exist, and so the effort would be to select only those portions of the ATP which verify the architecture, and to add tests as required to increase both coverage and confidence. Tests which extend MIL-STD-1750 to AN/AYK-15A requirements would need to be modified.

[illegible] Resources.

An experienced architecture verification person would be required to select and modify the portions of the ATP which concern architectural verification. The actual execution of the method can be performed by a relatively inexperienced person. A person experienced in architectural verification could be required for error analysis, although error analysis is probably a contractor's responsibility.

[illegible] Observations and Applicability to MIL-STD-1750

An advantage of the AN/AYK-15A ATP approach is that the programs have been designed and coded. Therefore, the only development necessary is to determine the subset of the tests required to meet MIL-STD-1750, delete AN/AYK-15A extensions, and to add tests as required by the architecture. The program is very thorough and exhaustive in its verification of the AN/AYK-15A implementation of MIL-STD-1750.

The disadvantage of the AN/AYK-15A ATP approach is that it requires a [illegible] because the core instruction set used is so small. [illegible] ATP, but is not required in an [illegible]. Also, special hardware is required [illegible] test. The generalization of [illegible] environment could prove difficult [illegible].

The AN/AYK-15A ATP method could provide a very thorough and cost effective approach to the MIL-STD-1750 verification. The complete AN/AYK-15A ATP is [illegible] dependent since it is designed to verify the AN/AYK-15A. However, the architectural verification [illegible] the ATP is [illegible] independent. The only application [illegible] the AN/AYK-15A ATP method is that the memory must be [illegible] the program.

[illegible] criteria and will be further [illegible].

6.2 RANDCM INSTRUCTIONS

6.2.1 Description

This is a statistical approach to the task of verification. The candidate computer is initialized to a known state. A block of random instructions is then generated and executed by the unit. All machine state results are then captured and compared against the expected results which are obtained through simulation. This process is continued until atistical evidence of completeness is acceptable.

There are two main programs required by the approach. A program to generate a block of random instructions and a valid simulator to generate the expected results. Control for the two programs is provided in a table of implementation dependent conditions.

A random seed is generated, saved for future recall and fed to the instruction stream generator which returns a variable length block of random instructions. This instruction stream is fed to the simulator along with the control table of implementation dependent conditions and the initial status of the machine. This is simulated upon the simulator, and the ending status of the sequence is returned for use as the expected values for the computer under test. The instruction stream is then run on the hardware, an interrupt is forced and the status variables captured for comparison with the expected results. The two results are compared and, if equal, the next test is generated. If not equal the stream is rerun to check for an intermittent failure. If the second run passes, the error is logged and the next test generated. If both runs fail, the last instruction in the stream is replaced by a NO-OP and the stream is rerun on both the hardware and on the simulator. The process is continued up the stream until a successful comparison is obtained. The last NO-OP is replaced by the instruction and the stream run on the hardware and on the simulator to capture expected and actual results at each stage of the stream. This information is then printed for use in diagnosing the failure.

6.2.1.1 Block Diagram

Figure 6-2 depicts a hardware block diagram for this approach.

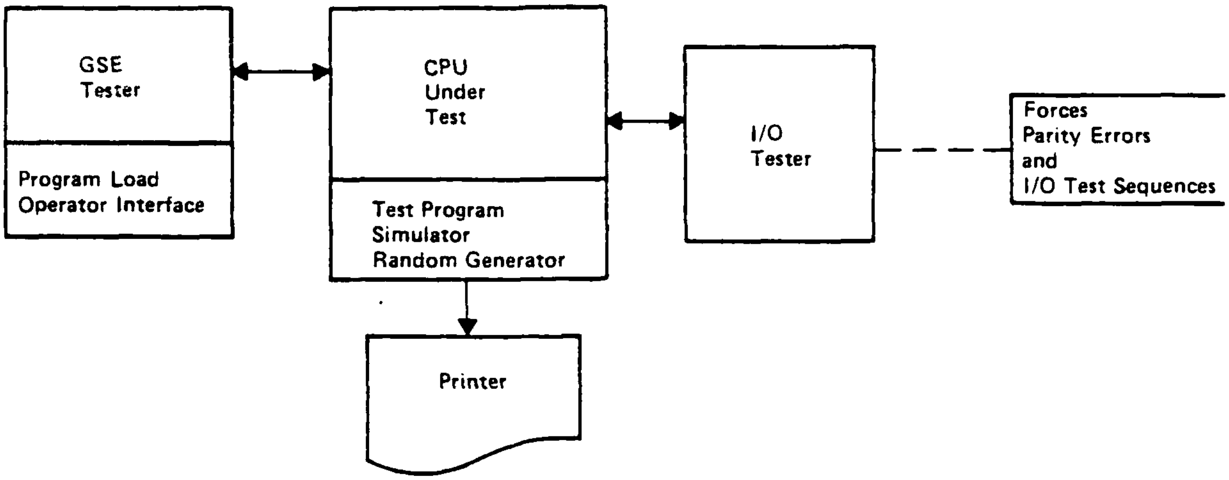


Figure 6-2. Random Instructions Block Diagram

6.2.2 Experience

IBM has used this method to check the completeness of the System 360/370 verification method and found that detection times were relatively short when known problems existed in the unit. Generally the error was found in the first 30 minutes of run time for a machine with the performance in the range of four million instructions per second.

Typically 1,000,000 test sets of randomly generated instructions from 1 to 32 instructions in length (average length of 10) can be run in a 8-hour period. Thus, it is expected that 10,000,000 instruction cases are executed in an 8-hour period for a four million instructions per second computer.

This method has been successfully used in all four stages of computer development:

1. by Engineering as a bring up tool

2. by Quality Assurance as a verification tool

3. by Manufacturing for final bcx checkout

4. by Field Engineering as a diagnostic and operational check.

6.2.3 Hardware Resources

The Randcm Instruction method requires a specialized tester to verify I/C, interrupt structure and concurrent cperations. This tester is designed fcr a particular computer and I/O configuration. Also, the ccmputer must have a special interface to the tester so that the tester can be controlled by the test prcgram. In addition, Ground Support Equipment is necessary for prcgram ccntrcl and error reporting.

6.2.4 Software Resources

A complete MIL-STD-1750 architectural simulator must be written to generate data for comparison. Additional advantages are gained if the simulator is written to run on the MIL-STD-1750 ccmputer itself; however, this may require prcvision for additional main storage. For example, I/O transfers are substantially reduced, but much more main storage is required for the simulator. Some type of an I/O tester is also needed for verification cf I/O related architectural features. A randcm instruction generator must be written to generate the instructions and compare the results.

6.2.5 Personnel

A person who is experienced in architectural verification and in simulation is required to write the prcgram for the MIL-STD-1750 simulator. A person with some support software experience and familiarity with the MIL-STD-1750 architecture is required to write the randcm instruction generator program. The actual execution of the test prcgram can be performed by a relatively inexperienced person. Error analyses could require a person with a high degree of training in architectural verification, although error analysis is probably a ccntractor's responsibility.

6.2.6 Observations and Applicability to MIL-STD-1750

The Pandom Instruction methcd provides an excellent method to verify ccmputer architectures. The method provides for a thorough testing of instruction sequence dependencies and other subtleties due to the random nature of the test cases. This approach also shows promise as a good method to improve confidence and completeness in other methods.

No detailed analysis of the architecture would be required to generate the test sequence.

A detailed analytical analysis has been unsuccessful at developing the criteria used for completeness. Engineering judgement has determined the number of instructions necessary for the verification tests to achieve a level of completeness; the random process can be terminated after 1,250,000 random instructions.

The Random Instruction is an excellent test technique; however, tests of boundary conditions (like a branch instruction in the last main storage location) would have to be added to completely test the MIL-STD-1750 architecture. The MIL-STD-1750 simulator required would have to completely represent the architecture.

The Random Instruction method does not contain any vendor dependencies in its test cases. The only application dependency in the Random Instruction method is that the main storage must be large enough to contain the program. If a self-hosted simulator is used in this method the main storage requirement could become quite large.

This approach passes the defined pass/fail criteria and will be discussed further in Section 7.

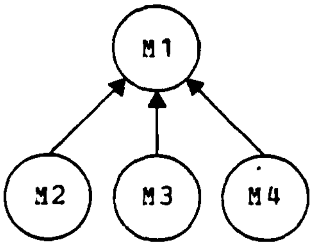
6.3 ANALYTICAL APPROACH

The following text is a summary of a very complex approach and requires a background in this field. Further information is available from appropriate documents located via the Bibliography (listed under Carter).

6.3.1 Description

This method consists of using an analytical method called Symbolic Execution to verify that the hardware implementation of the computer meets the architectural specification. The hardware implementation in the form of microcode or finite state information is translated into a machine readable format for analysis by a program which verifies by Symbolic Execution that it conforms to the architectural specification. The specification must also be supplied in some machine readable format.

In this approach two definitions are required; one for the architecture specification and the other for the machine implementation. Both must be given in the architectural description language "Language for Symbolic Simulation" (LSS). These definitions consist of a facility vector describing machine components and a ,decision tree describing the control structure. The facility vector specifies the various components which can be observed or manipulated by the machine. The operations upon the facility vector components are specified by macro routines written in LSS and the order of macro application is determined by the control tree operation algorithm. (That is, each decision tree is composed of macros.) For example, in the following control tree segment



Macros M2, M3 and M4 must be carried out before M1 can begin. Macros M2, M3 and M4 may be performed in any order. When M2, M3 and M4 have been performed, M1 is initiated. The facility vector components are treated by LSS as APL-like variables with specified dimensions.

Because the two facility vectors (specification and implementation) generally have different components, to prove correctness it is first necessary to specify a relation between them. One machine action simulates another if everything that the first machine can do, the second can do, though possibly in a different way. The states of correspondence for the symbolic simulation are determined by pairs of tree control structures called simulation control points, and predicates. Predicates describe the conditions that the initial values of the facility vector must satisfy in order to be in a particular state. The states of correspondence specify relations between the components of the respective facility vectors at these pairs of points. The set of all machine states pairs and their relation forms the simulation relation.

Proofs of equivalence generally proceed as follows. For each pair of corresponding machine states in the simulation relation we must:

1. Assert that the simulation conditions corresponding to each component hold.

2. Run each abstract machine, following all paths at branch points and performing symbolic computation until another machine state is reached.

3. Verify that the new pair of states corresponds to a component of the simulation relation.

4. Prove that the simulation conditions of this component hold.

In such an execution analysis for proving equivalence, facility vector parameters have as initial values either symbolic constants (symbols representing unknown but fixed values), or values determined from the simulation relation. When symbolic constants are encountered in expressions being computed in assignment statements as part of Symbolic Execution, the value assigned is an expression involving operators and symbolic constants. This expression is always simplified combining like parameters.

When symbolic constants occur in predicates evaluated to determine possible branches, a single flow of control may not be able to be determined, since the predicate may evaluate to a Boolean expression involving symbolic constants, which can not be evaluated to "true" or "false". In such a case all possible logically independent results of evaluating the predicate must be considered. The program doing the symbolic evaluation will now generate a subgoal for each independent result, add a predicate expressing the truth of the result to the predicate list, and simplify the result. One path is taken; the remaining paths will be followed later. (If the predicate involving symbolic constants evaluates to true or false, then the preceding multipath analysis is not required in that only the one path analysis need continue.)

After generating a series of sets of subgoals, a simulation control point for this level is reached. Now it is required that the other machine is run until a simulation control point for that level is reached.

At this point, a comparison is made between the specification result and the implementation result. That is, the program verifies that the pair of control points reached defines a component of the simulation relation. The values of the two facility vectors are substituted into the simulation conditions of this component and, using the predicate list, all of these conditions are proved to hold.

This process is repeated for each of the remaining branches of the tree. In the process a complete goal tree is formed. When a complete goal tree is formed, the analysis has been successful.

It may occur that a correspondence can not be proved and thus, the goal tree cannot be completed. Then an error must be sought in the code or in one of the descriptions. In addition the occurrence of an unexplained branch will signal the presence of an error.

6.3.1.1 Example

The following example is provided for further illustration of the concept of symbolic execution:

if $X<1$ then $Y := 1-X$ else $Y := X-1$

Suppose we wish to prove that after the statement is executed $Y = |X-1|$. The object of the verification is to construct a proof tree. A node in the tree represents a class of states of the system at one point in time. The root describes all the initial states in which our program can start execution -- one state for each pair of initial values for X and Y. The leaves of the proof tree represent all the possible final states -- for each leaf we have to prove that the states represented by the leaf satisfy $Y = |X-1|$. A branch in the proof tree represents a computation path.

In our example the tree is first initialized to the following root:

1. Facility vector:

   X contains v
   Y contains w (where v and w are some arbitrary symbols)

2. Predicate: empty

3. Control point: before the if statement. The root represents all those states where X and Y contain some arbitrary values v and w, which are not constrained by anything (predicate list is empty), and where the if statement is just about to be executed.

Symbolic execution is used to construct a tree of nodes representing states reachable from the root by executing the given program. The first statement is an if statement. Since our current node (the root) represents states where $v<1$ as well as states where $v\geq 1$ we must split the execution into two cases (i.e., two sub-goals). The first subgoal represents those states where $v<1$. It has the same state vector as the root, its predicate is $v<1$, and control is just before the assignment Y := 1-X. The second subgoal represents those states where $v\geq 1$. Again it has the same state vector as the root, its predicate is $v\geq 1$, and control is just before the assignment Y := X-1. Now we choose one of these new leaves as the current node, say, the first subgoal, and proceed. The assignment Y := 1-X is executed by changing the contents of Y to the expression 1-v and advancing the control point. Since the control has reached the end of the program we are supposed to prove the assertion $Y = |X-1|$. Or, more exactly, it must be proved that the assertion holds in every state represented by

1. Facility vector:

   X contains v
   Y contains 1-v

2. Predicate: $v<1$

3. Control point at the end of program (actually irrelevant for correctness).

Therefore we must prove that the predicate $v<1$ implies the assertion with X and Y replaced by their contents. And that is the verification condition

$$v<1 \text{ implies } 1-v = |v-1|,$$

which is clearly true. In an analogous way we obtain from the second case the verification condition

$$v\geq 1 \text{ implies } v-1 = |v-1|,$$

which is also true. In the above example our simulation relation would consist of two components. In the first component the stopping point identifies the beginning just before the if statement; the associated assertion is true (i.e., no assumption about initial values). In the second component the stopping point identifies the end just after the if statement; the associated assertion is $Y = |X-1|$. Thus, the goal tree is completed.

6.3.1.2 Functional Diagram

A functional diagram does not apply for this approach.

6.3.2 Experience

This method is currently being investigated by Dr. W. Carter at IBM's Research facility in Yorktown, New York.

6.3.3 Hardware Resources

This method does not require any specific hardware resources. (However, it does presuppose a computer is available with support of the LSS system.)

6.3.4 Software Resources

Each vendor's implementation as well as the architecture specification would have to be described in a machine readable format such as LSS (Language for Symbolic Simulation). Also each vendor's implementation would require a different definition of the simulation relation.

6.3.5 Personnel

This method requires personnel who are highly trained in the area of simulation, and are familiar with architecture description languages such as LSS.

6.3.6 Observations and Applicability to MIL-STD-1750

This approach shows much promise for providing a very complete design verification of the architecture. It would also allow designs to be verified before hardware was built. Since each of the possible paths in the proof tree are followed for both the architecture specification and the hardware implementation, the method will generally catch subtle second order effects.

This could be a very high cost approach since each vendor's implementation would have to be described in a machine readable format. Also, the risk in using this approach is very high because verification using this approach has not yet been demonstrated. Another problem is that this method verifies that the abstract definition of the hardware conforms to the formal specification. It does not validate that the intended functions of the architecture are built into the hardware. The actual hardware would still need to be checked for wiring errors, noise, timing races, etc. Certification would require some type of further testing.

Due to the nature of the tools required, this method is judged not feasible at this time, and will not be considered for further analysis.

6.4 ARCHITECTURAL VERIFICATICN PRCGRAM - SYSTEM 360/370

6.4.1 Description

The IBM System/360 architecture verificaticn approach consists of a set cf programs which allow test cases tc be written in a procedural language. These test cases are stcred on magnetic tape. A supervisor, which is resident on the ccmputer whcse architecture is to be validated, reads the test cases frcm the tape, executes them, and checks for correct results. Any errors found are printed out.

A test case is written to test each item cf the architecture. Each test case is similar to what an engineer would normally enter by hand tc test the same function. Each test case contains the setup and expected results for testing one characteristic of cne instruction. In some cases, primarily I/O, more than one instruction is required fcr proper execution. Each test case is written in such a way as to test cnly one function at a time. The supervisor program automates the process by setting up the system, executing the test case, and ccmparing the expected tc the actual results.

The supervisor is a 4K stand-alone program which requires a printer, console, and some type of mass storage. In testing I/O channels, extensive use is made cf a program controlled I/O Adapter to force parity errors and specific I/O line sequences. This supervisor reads test cases from tape, and then sets up all general purpose, floating pcint, and control registers, and low core and main storage as specified. It then executes the test case instructicn and initiates ccmparisons with the expected results. Comparisons include any storage areas specified; all general purpose, control, and floating pcint registers; and all of low core frcm addresses 000-1FF. Input test cases are bypassed if they require features that are not on the machine being tested. Unless ctherwise specified in the test case, all instructions are repeated via the EXECUTE instruction to test this feature of the System 360/370 architecture.

Fcr successful cases, the test case number is printed out for documentation purposes. When an error is detected, the supervisor prints out both the expected and the actual results for the failing comparison. Only the error data is printed and, in crder to analyze the problem, it is necessary to refer to the test case listing.

The verification program is initiated by loading the supervisor program frcm a card reader or frcm tape. When loading is complete, the system enters a wait state. The user console is used to start execution. The console can also be used to input additional test cases when desired.

6.4.1.1 Block Diagram

Figure 6-3 depicts a hardware block diagram for this approach.

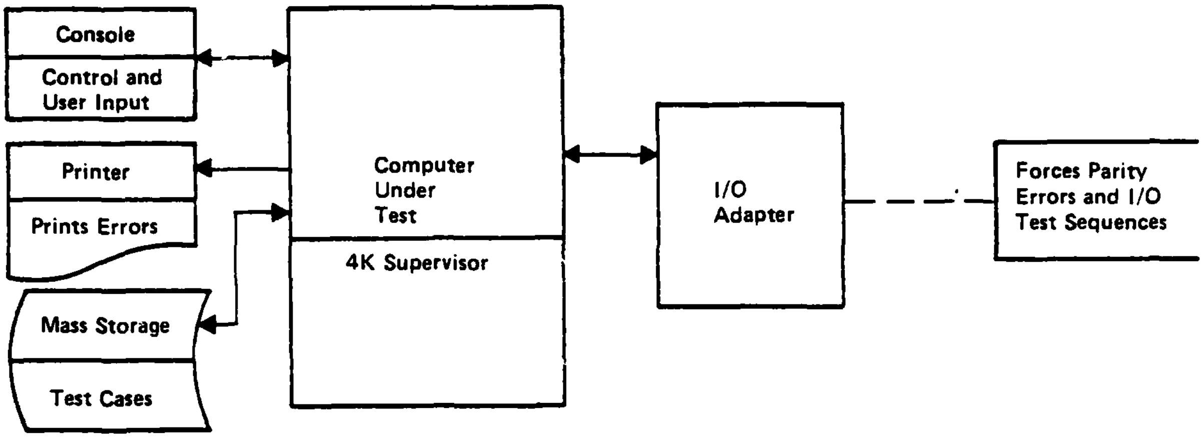


Figure 6-3. IBM System/360 Architecture Verification Elock Diagram

6.4.2 Experience

This method is used by IBM to validate the System/360 architecture on all IBM 360/370 computers. This method is extremely successful as demonstrated by success of the System 360/370 architectural compatibility.

6.4.3 Hardware Resources

This method requires the use of an I/O Adapter, which simulates I/O channels, to permit the testing of the 360/370 I/O channels and CPU instructions which affect the channels. A direct I/O wrap cable is also required to verify the direct control feature. A console, printer and some type of mass storage are also required to execute the test cases.

6.4.4 Software Resources

This method requires a control program (which is run on the computer under test), the test cases, and support programs to update the test cases.

6.4.5 Personnel

A person who is experienced at architectural verification is required to write the test cases. The actual execution of the test cases can be performed by a relatively inexperienced person. A person experienced in architectural verification could be required for error analysis, although error analysis is probably a contractor's responsibility.

6.4.6 Observations and Applicability to MIL-STD-1750

This approach relies on a large number of test cases and experience to provide for thorough testing of the architecture. If a sufficient number of test cases are used, it can provide a very thorough method to verify an architecture. The evidence of its success is the high transportability of utilities and user programs among members of the System 360/370 family. Due to the nature of the test cases and the structure of the supervisor, additional tests could easily be added to the verification program.

This method's greatest drawback is that the test cases must be generated manually. The appropriateness of the test cases generated depends on the insight of the person writing them. During the initial use of this method, many features of the architecture may remain unverified, but as problems are found and test cases to uncover these problems are added, the thoroughness of the approach increases. After sufficient time has passed, this approach can mature to provide a very thorough method of verifying an architecture.

This method allows extensive testing in a "quiet" environment. It does not provide a facility to test interaction between channels or between channels and the CPU. This method tests all aspects of each instruction with the exception of data dependency. An attempt is made, however, to use data that either has caused problems in other systems or is suspected of being critical. Since a test case for every bit combination for each instruction would be astronomical, any data dependency situations that are suspected are coded up and added as they occur.

This method could be used to provide a vendor and application independent verification of the MIL-STD-1750 architecture. The greatest effort required to implement this method would be in designing the test cases. Initially, test cases would probably be gathered from vendors' ATPs, FTPs, and Diagnostics. As deficiencies were found in the procedure, test cases would be added to correct them. Additional test equipment would have to be built to verify any I/O defined in MIL-STD-1750. Also, some method of listing any failures and the data associated with them would have to be provided. This approach passes the defined pass/fail criteria and will be considered further in Section 7.

6.5 DIAGNOSTIC

6.5.1 Description

The Diagnostic approach to architecture verification is to make use of the individual manufacturer's diagnostic programs. The verification program is created by taking the diagnostic programs and removing hardware implementation dependent tests from them and using only the remaining parts that are functional in nature. Because of this removal of hardware dependent tests, some additional tests may need to be written in order to completely verify the architecture.

The Diagnostic method consists of taking each manufacturer's diagnostic program for his computer and combining them into one architecture verification program. The individual diagnostics cannot be taken in total, since they may contain implementation dependent tests. That is, various manufacturers may have microcoded Built-In Test Equipment (BITE) tests or additional hardware features, (not defined in MIL-STD-1750) to increase the diagnostic capability of the computer. Although all computers are MIL-STD-1750 compatible from a user's viewpoint, they are not identical from a diagnostic viewpoint since the hardware implementation is different. Therefore, the manufacturer's diagnostics must have all non-MIL-STD-1750 hardware implementation dependent differences removed from them. Once this is done, the diagnostics will run on any MIL-STD-1750 computer. However, the removal of the implementation dependent tests decreases the coverage of the resulting Diagnostic architecture verification program. Additional tests may then have to be added to restore the coverage to an acceptable level.

6.5.1.1 Block Diagram

Figure 6-4 depicts a hardware block diagram for this approach.

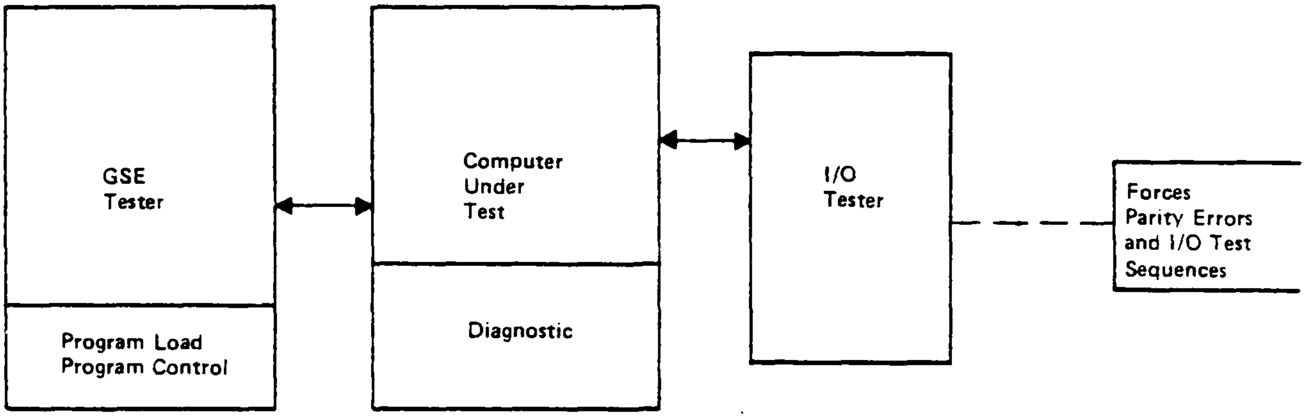


Figure 6-4. Diagnostic Block Diagram

6.5.2 Experience

The architecture of the Advanced Signal Processor (ASP), which was developed by the IBM Federal Systems Division in Manassas, Virginia, was verified by using the Diagnostic approach. This method was also used by the Army/Navy Computer Family Architecture project in verifying the PDP-11/70 computer architecture.

6.5.3 Hardware Resources

Since various manufacturer's have added microcode and/or additional hardware to augment the diagnostic capability of their computer, it would be useful to use those features in verifying the architecture. For example, if a manufacturer implemented a means of setting bits in the Interrupt Pending Register, this feature could be used to assist in verifying the interrupt priority structure. However, to be of use in architecture verification, the feature would have to be implemented by all manufacturers. This means that the feature would have to become part of MIL-STD-1750. Therefore, this method might require extensions to MIL-STD-1750 rather than special additional equipment.

6.5.4 Software Resources

The software resources required by this method are the diagnostic programs designed by each vendor. These programs would be used to produce the Diagnostic verification program.

6.5.5 Personnel

A person experienced in architectural verification with some understanding of diagnostics would be required to design the resulting program. The actual execution of the method can be performed by a relatively inexperienced person. A person experienced in architectural verification could be required for error analysis, although error analysis is probably a contractor's responsibility.

6.5.6 Observations and Applicability to MIL-STD-1750

A disadvantage with this method is that it relies on the diagnostics to be written in a functional, rather than a hardware-oriented method. This could result in the diagnostics becoming much larger and more costly than might be justified. Combining functional diagnostics from several vendors may or may not provide complete verification and would certainly provide several redundant tests.

The Diagnostic method provides a good source of data to test the critical points of algorithm implementation by various vendors. However, significant effort would be involved to remove implementation dependent tests from each vendor's diagnostic and then to combine the various diagnostics into one Diagnostic verification program. Also, this resulting Diagnostic program would have to be evaluated to determine if removing the implementation dependent tests seriously compromised its usefulness. If this was found to be the case, additional effort would be required to formulate additional tests to provide better coverage of the architecture.

The Diagnostic method starts out inherently vendor dependent. Each Diagnostic program has test cases which make use of implementation dependent hardware features and knowledge of the implementation to reduce the number of test cases. However, these dependencies could be removed to provide the final Diagnostic verification program. The only application dependency in the resulting Diagnostic method is that the memory must be large enough to contain the program.

If extensions were added to MIL-STD-1750 to include BITE facilities, it might be possible to use vendor's diagnostics without extensive modification.

This method passes the defined pass/fail criteria and will be further discussed in Section 7.

6.6 FUNCTIONAL TEST PROGRAM

6.6.1 Description

The Functional Test Program (FTP) apprcach verifies the architecture by performing functional level testing of the ccmplete instruction repertoire, the main storage, the I/O, the interrupt structure and the concurrent functions. The instructicn repertoire is verified by exercising all instructions and their formats as defined in the architectural specification. The architectural specification describes each function to the level of detail that must be understood in order to prepare an assembly language program that relies on that function. Instructions are exercised using all addressing formats, registers, interrupts, and condition codes. The main storage, I/O, and interrupts are also tested by exercising their functions as defined in the architectural specificaticn manual. Test equipment is required for the man/machine interface (contrclling, loading) and causing interrupts, I/O and exception sequences.

The FTP method uses a center out apprcach to instruction set verification. First, a core set of instructions are verified. This core set is then used to verify the remaining instructions. Although all addressing formats, registers, interrupts and condition codes are exercised, the testing is not exhaustive for all data patterns. The necessity of exhaustive testing is eliminated by kncwledge of the hardware implementation. Also, because the FTP is used to debug the hardware and for environmental testing, it must be of a manageable size and cycle in a short period cf time. This, therefore, precludes the use of exhaustive testing.

The method used for main storage testing is dependent on main storage usage. Normally, the read/write porticns cf the main storage test do not check the portions cf the main storage in which the test routine and supervisor reside. These parts cf memory are checked by checksums. This is due to the restriction that the complete FTP be contained in the main storage at all times.

The verification of I/O, interrupt structure and concurrent function is aided by the use of specialized testers. The tester stimulates the external inputs and the FTP verifies the proper operation of the computer. For external outputs, the FTP generates the outputs and they are stored in the tester for later analysis by the FTP. This requires that there be an I/O tester interface between the tester and the computer under test. The tester is also able to generate a number of error conditions so that proper operaticn of the computer can be verified.

The FTP also exercises all Built-In-Test Equipment (BITE) hardware for proper operation and to detect any hardware failures.

Ground Support Equipment is required for program loading, control and error reporting.

6.6.1.1 Block Diagram

Figure 6-5 depicts a hardware block diagram for this approach.

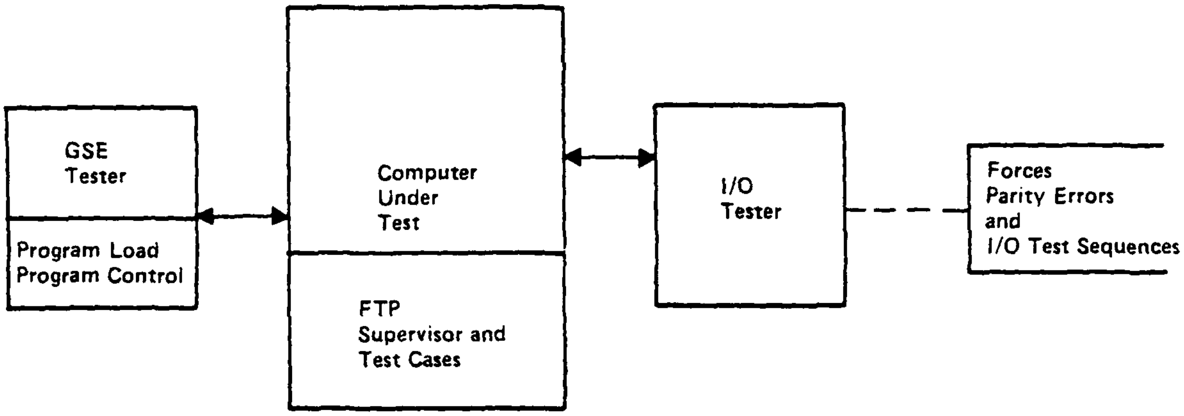


Figure 6-5. FTP Block Diagram

6.6.2 Experience

The IBM Federal Systems Division has made extensive use of this method for verifying many computer architectures. The FTP is used as a development tool to verify that the hardware implements the architecture. In addition, the FTP is used as part of the Customer Acceptance Test Procedure, as a reliability demonstration tool, and as an engineering evaluation tool during hardware bring-up.

6.6.3 Hardware Resources

The FTP method requires a specialized tester to verify I/O operations, the interrupt structure and concurrent operations. This tester is designed for a particular computer and I/O configuration. Also, the computer must have a special interface to the tester so that the FTP can control the tester. In addition, Ground Support Equipment is necessary for program control and error reporting.

6.6.4 Software Resources

The software required for this method is the Functional Test Program itself. Since the test cases are imbedded in the program, no special facilities are required to add or update test cases. However, this makes the addition or updating of test cases more time consuming because parts of the FTP would have to be reassembled and relinked.

6.6.5 Personnel

A person who is experienced at architectural verification is required to design the FTP. The actual execution of the FTP can be performed by a relatively inexperienced person. Error analysis could require a person with a high degree of training in architectural verification, although error analysis is probably a contractor's responsibility.

6.6.6 Observations and Applicability to MIL-STD-1750

This approach has been used by the IBM Federal Systems Division to verify its military computers. If the implementation of the architecture is known, then use of the FTP method for verification yields a high degree of confidence at a reasonable cost.

In IBM's Federal Systems Division experience, about 90 percent (coverage) of the hardware will be exercised if only the architectural specification is used to develop verification tests. The reason is that certain instructions can be implemented in different ways, each of which yields correct results but which might require different types of tests to see that no implementation error has been made. The IBM Federal Systems Division has found that FTPs cover better than 99 percent of the hardware after utilizing implementation information. If a general purpose verification tool is to be developed like an FTP, more exhaustive testing will be required given that the methods of implementation cannot be predetermined.

The FTP method can provide a very cost effective method to verify a computer architecture. However, to be effective in a multiple vendor environment, tests must be enhanced to make the testing more exhaustive since the hardware implementation is no longer known. An existing FTP may contain some vendor dependencies in its test cases. For example, the use of a Diagnostic instruction or BITE hardware facilities would make an FTP vendor dependent. However, these test cases could easily be removed. The only application dependency in the FTP method is that the memory must be large enough to contain the program.

This method passes the defined pass/fail criteria and will be given further consideration in Section 7.

6.7 INSTRUCTION SET PROCESSOR

6.7.1 Description

The Instruction Set Processor (ISP) approach consists of first specifying the architecture of a machine in a Computer Hardware Description Language such as Instruction Set Processor Specifications (ISPS). This ISPS description must include all of the information normally found in the architectural specification manual. The ISPS specification is then used as input to a program which automatically produces test cases for a verification program. This verification program is then executed on the computer to verify conformity.

To date, most Computer Hardware Description Language research has used the ISP family of languages in the areas of description, simulation, hardware synthesis, software synthesis and verification. Although other Computer Hardware Description Languages could be used for this approach, we will focus on the latest version of ISP (ISPS).

ISPS was developed by Carnegie - Mellon University to describe precisely the program level of a computer. The ISPS description provides a standard, unambiguous description that can be used to specify future software/hardware development and this description also provides a vehicle for defining the required information used for testing the architecture. An ISPS description defines the storage elements and sequences of operations of a processor. An ISPS description consists of a block of storage declarations and sequences of register or control transfer operations.

The storage elements include storage available to the programmer (e.g., general registers and main storage) and local storage for the processor (e.g., internal registers like program status). The sequence of operations includes sequencing control of the data operations, and specifies how the processor fetches, decodes, and executes instructions. Storage elements are organized information structures; for example, a main storage might consist of a number of words, each of a number of characters, each of a number of bits, or a register might consist simply of a number of bits. Storage elements are defined in ISPS by a name and description of their structure.

Main storage is referenced, other than in storage declarations, by its name, qualified by an index if it is a multiword main storage. Multiword main storages must be accessed using an arithmetic expression. Main storage references can be further qualified with a range of bits to be selected.

Register transfers are used to describe the data operations on storage elements. Since most operations in a computer result in modifications of bits in storage elements, each action in a data transfer sequence takes the form:

storage-expression<---data-expression

The data-expression describes the transformation of information and the bit pattern that is to be placed in the storage element designated by the storage-expression. Data operators in ISP include transfer, relational, arithmetic, Boolean, shift and concatenation operators. All data operators assume unsigned binary representation.

The evoking of actions can be controlled by conditional actions of the form:

(IF condition => action-sequence)

where the condition (a Boolean expression which is either false or true, where false is defined as 0 and true as any non-zero value) describes when the action-sequence will be evoked, and the action-sequence describes what transformations take place on what elements.

The capability to select one of a list of alternative statements to be executed is provided by the DECODE statement:

DECODE expression = statement-list

where the value of the expression is interpreted as an integer and used to select one of the alternative statements in the statement-list. The alternative statements are not considered to be concurrent activities, but are a list of statements where the ith statement is executed if the value of the expression is equal to i.

The information contained in the ISP description is used as input to a test generator program which determines, from the storage declarations and register control transfer operations, what tests are required for each instruction or operation, and then generates assembler level code covering these test requirements.

The test requirements consists of checks for the following:

The functional results
Values in registers used and unused
All required interrupt mechanisms
Main Storage locations, both used and related
Condition codes, set or unaffected
Error indicators
Order of occurrence of predicted events
Possible interference due to simultaneous I/O and/or interrupts.

Sufficient data patterns would have to be either provided to the test generator as input or be created by the generator.

The generated test program is then executed to verify that the ccmputer meets the descripticn.

Grcund Support Equipment is necessary for prcgram loading, control and error reporting.

6.7.1.1 Block Diagram

Figure 6-6 depicts a hardware diagram for this apprcach.

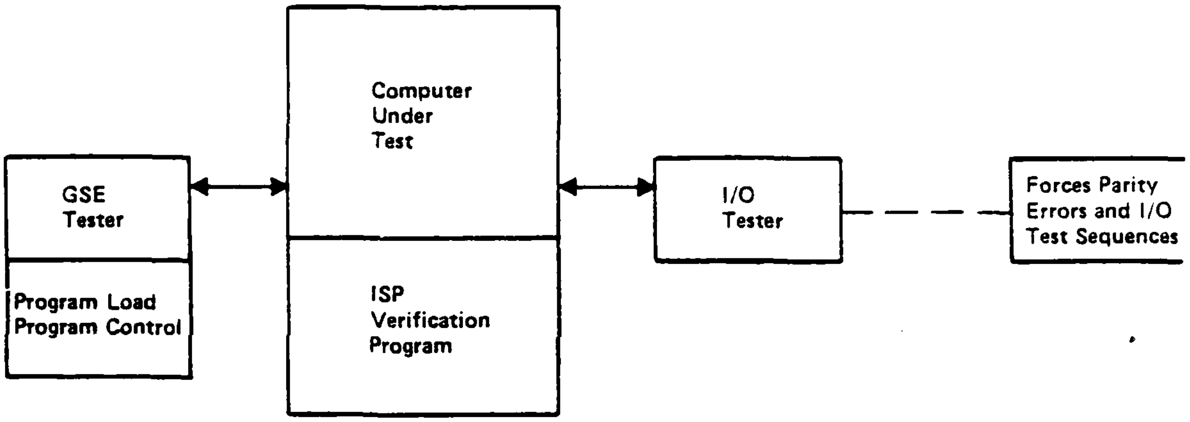


Figure 6-6. ISP Blcck Diagram

6.7.2 Experience

ISPS has been widely investigated by the research ccmmunity as a vehicle fcr computer hardware description. IBM is currently developing automatic test generaticn methods to verify architectures because they show much promise for future use. At the present time, IBM is developing PL/I programs which generate assembler level code to test limited sets of instructions for the MIL-STD-1750 and NATO AWACS architectures.

6.7.3 Hardware Resources

The ISP method requires a specialized tester to verify I/O operations, the interrupt structure and concurrent CPU and I/O operations. This tester is designed for a particular computer and I/O configuration. Also, the computer must have a special interface to the tester so that the tester can be controlled by the Test program. In addition, Ground Support Equipment is necessary for program control and error reporting.

6.7.4 Software Resources

A complete and unambiguous description of MIL-STD-1750 must be written in ISP. The software required to generate the Test program is a high order language program capable of generating all necessary tests from key words and symbols in the ISP description; but it doesn't exist and needs to be written.

6.7.5 Personnel

A person who is experienced in architectural verification and compiler development techniques is required to write the program for the test generator. The actual execution of the test cases can be performed by a relatively inexperienced person. Error analysis could require a person with a high degree of training in architectural verification, although error analysis is probably a contractor's responsibility.

6.7.6 Observations and Applicability to MIL-STD-1750

This approach shows much promise for providing a very complete verification of the architecture, but still has the "all possible combinations" limitation. However, at this time, it has a high risk factor associated with it, because all of the tools needed for its implementation have not been completely developed. The benefit of this approach is that it could be one of the most rigorous and thorough verification methods.

A good deal of benefit could be gained by using a Computer Hardware Description Language to define the MIL-STD-1750 architecture even if auto test generation was not attempted.

A difficulty with the ISP method is that Computer Hardware Description Languages are still research tools. The success of the method will depend on the maturity and quality of the language used. ISPS is known to have some inconsistencies and ambiguities stemming primarily from the lack of a clear, precise, formal semantic definition of the language.

Another problem is that automatic test generation programs are also in the research stages and have not yet been used to generate I/O or interrupt tests.

MIL-STD-1750 would need to be defined in terms of ISP and the test generation program developed for the MIL-STD-1750 architecture.

The testing techniques used in the ISP approach are similar to those used in FTP and AVP approaches. The main difference is that the architectural analysis effort is applied in writing a high order language program which can determine, from the architectural description in ISP, all the tests that are needed to validate the architecture. This approach is not considered to be feasible at this time since the tools required are still in the research stage. For this reason the ISP method fails the pass/fail criteria and it will not be considered for further analysis.

6.8 LOCKSTEP

6.8.1 Description

The Lockstep approach consists of comparing the computer under test against one which is defined as conforming to the architecture. The procedure consists of running an identical program on computers which are synchronized to each other. Synchronization can be at any timing level or machine state. The test equipment then verifies that both computers produce the same results at each synchronizing point. Since one of the computers is defined as meeting the architecture, if both produce identical results when running the same program, then the second computer also meets the architecture.

For a complete understanding of the lockstep method, the facilities provided by the Trace interrupt must first be understood. The Trace interrupt, if enabled, will generate an interrupt after the execution of each instruction. This enables the software to Trace the execution of the program. In addition to generating the interrupt, the hardware also stores data concerning the state of the computer into fixed storage locations. The data saved includes: the instruction counter, storage address register, condition status, and the contents of all general registers. This data can then be used by the software to trace the execution of a program. The data stored by the Trace interrupt concerning the state of the computer is the most important part of the Lockstep method.

The Lockstep method consists of two computers, one previously verified by some method, which is called the "golden" computer and runs the tests, and the other computer which is the one to be verified. The "golden" computer loads the test to be run from the diskette and informs the computer under test via a serial channel which test to load from its own diskette. The test is run on both computers and the Trace results are saved in a buffer after the execution of each of the test case instructions. Once the test case is executed, the "golden" computer then reads the buffer from the computer under test via the serial channel. It then compares the results obtained from the computer under test with its own results to see if the Trace data is identical. Any differences in the Trace data indicate an error and the miscomparing data is printed out.

Both computers have a real-time operating system running in them. Executing under this operating system is the program which selects the test cases to be run, communicates with the other computer, and compares the Trace data.

6.8.1.1 Block Diagram

Figure 6-7 depicts a hardware block diagram for this approach.

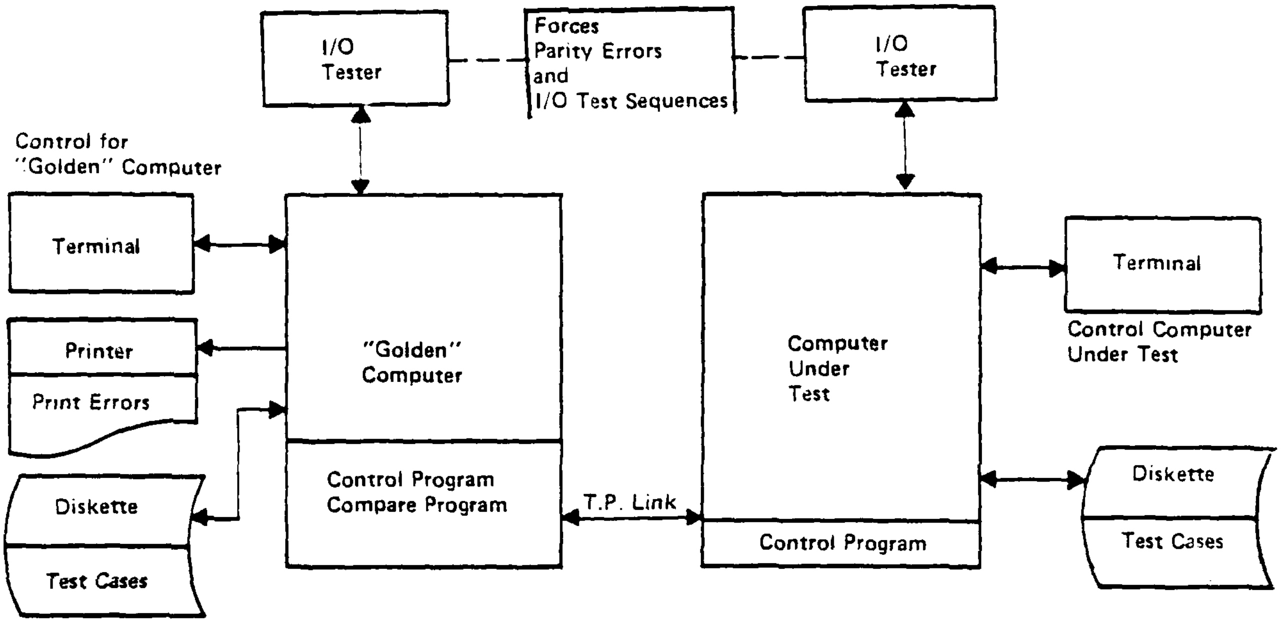


Figure 6-7. Lockstep Block Diagram

6.8.2 Experience

The IBM Corporation has used this method to verify that all Series/1 computers comply to the Series/1 architecture. It was also used by the IBM Federal Systems Division on the Verdin Correlator and Space Shuttle I/O Processor.

6.8.3 Hardware Resources

This method requires a "golden" computer as well as hardware within the computer under test to implement the Trace facility. This Trace hardware would not be difficult or expensive to design in a new computer, but it could very difficult to add on to an existing computer. In addition, some type of hardware tester is needed to check I/O. Also, the computer must have facilities for attaching a CRT terminal, some type of mass storage (e.g., diskette), and a serial channel.

6.8.4 Software Resources

This method requires a real-time operating system to perform all communication with the peripherals. A control program, and the test cases to be executed are also required.

6.8.5 Personnel

A person who is experienced in the design of real-time operating systems is required to design the operating system and the control program. A person experienced in architectural verification is required to design the test cases. The actual execution of the method can be performed by a relatively inexperienced person. A person experienced in architectural verification may be required for error analysis, although error analysis is probably a contractor's responsibility.

6.8.6 Observations and Applicability to MIL-STD-1750

The hardware Trace facility allows the comparisons between the two computers to be very detailed and thorough. This means that the verification procedure will have a high degree of confidence. Also, once started, the procedure is completely automated, and so an operator need not be present when the test is being run.

The major disadvantage with this method is that the first computer (golden) must be verified by some other method. Also, the test cases must be generated manually. This means the quality of the test cases is determined by the person developing them. If the Trace interrupt facility is not part of the architecture, the addition of this facility to the architecture could be a major problem.

The test case generation and execution part of this procedure is similar to the System/360 AVP method. The main difference is the TRACE interrupt facility and the use of a second computer to compare results. It is possible to modify this method so that only the computer under test is needed. Instead of comparing data to that generated by the "golden" computer, the "golden" computer's data can be stored on an external device (diskette) and the computer under test can compare its data to the stored data. This would require a larger mass storage device. The Lockstep method does not contain any vendor dependencies in its test cases; however, unless the Trace interrupt facility is standardized in the architecture, vendor dependencies could arise in the control of the test program. The only application dependency in the Lockstep method is that the memory must be large enough to contain the program.

This method passes the defined pass/fail criteria and will be given further consideration in Section 7.

6.9 SUMMARY

The eight approaches described above are summarized in Table 6-1. They can be grouped into four different generic types. These types are Functional, Random Instruction, Lockstep, and Analytical. The functional type consists of the Functional Test Program, Architectural Verification Program, Acceptance Test Program, Diagnostic Program, and the ISP method.

The Functional type of verification procedure consists of a program which verifies the architecture by executing a number of test cases which test the architecture at a functional level. The origins of the program vary depending on the method used to generate the functional type test. The Functional Test Program (FTP) method starts with the FTP which is used to debug the hardware and generalize it. The Architectural Verification Program (AVP) method uses the AVP which is more general and more exhaustive then the FTP. The Acceptance Test Program (ATP) method uses the ATP which is usually somewhere in between the FTP and AVP in generality. The Diagnostic Program starts with the diagnostics for the hardware and attempts to generalize them and remove any implementation dependencies from the test. The ISP method attempts to generate functional tests automatically from a detailed description of the architecture. The ISP method is an automated AVP.

The Random Instruction type consists of automatically generating random sequences of instructions, executing them, and verifying that the proper results are generated. The expected results are determined by simulating the random sequence of instructions on a simulator. The main effort in this approach is to design a simulator which models the architecture as closely as possible. The architecture must be modeled with sufficient accuracy so that the simulator and actual hardware give identical results.

The Lockstep type consists of running a functional type test program on two computers and comparing the results of the test run on the two computers. To aid in the testing, a hardware trace facility is added to the computers. This facility allows all pertinent data concerning the state of the computer to be saved after the execution of each instruction. This state information is saved after the execution of each instruction of the test case and compared between the two computers. The main effort in this approach is the implementation of the trace hardware and designing and coding the functional type test program.

The Analytic Research type consists of converting the architectural description and the implementation (microcode and/or logic diagrams) into symbolic language descriptions and symbolically executing both to verify that they produce the same result. This method relies on complete descriptions of both the architecture and the implementation.

Table 6-1. Summary of Verification Approaches

	GENERIC TYPE	TESTS BOUNDARY CONDITIONS	TESTS HIGHER ORDER EFFECTS	COMPLEXITY	TEST ASSUMPTIONS	TESTS HARDWARE	INSTRUCTIONS USED	HARDWARE KNOWLEDGE	ARCHITECTURAL ANALYSIS EFFORTS
ATP	FUNCTIONAL	YES	NO	AVERAGE	NOTHING WORKS	YES	FEW	YES	TEST CASES
RANDOM	RANDOM	NO	YES	HIGH	EVERYTHING WORKS	YES	ALL	NO	SIMULATOR
ANALYTICAL	ANALYTICAL	YES	YES	HIGH	————	NO,DESIGN	——	YES	DESCRIPTION/ SYMBOLIC SIMULATOR
AVP	FUNCTIONAL	YES	NO	AVERAGE	EVERYTHING WORKS	YES	ALL	NO	TEST CASES
DIAGNOSTIC	FUNCTIONAL	YES	NO	AVERAGE	NOTHING WORKS	YES	FEW	YES	TEST CASES
FTP	FUNCTIONAL	YES	NO	AVERAGE	NOTHING WORKS	YES	FEW	YES	TEST CASES
ISP	FUNCTIONAL	YES	NO	HIGH	EVERYTHING WORKS	YES	ALL	NO	DESCRIPTION/ AUTOMATIC GENERATOR
LOCKSTEP	LOCKSTEP	YES	SOME	AVERAGE	EVERYTHING WORKS	YES	ALL	YES	TEST CASES

The main effort in this approach is in converting to the symbolic language and executing it. It is important to realize that this method verifies the intended implementation (the logic) not the actual hardware. Also, this method could be used for verification before the actual hardware is built.

6.9.1 Pass/Fail Evaluation

The approaches were evaluated using Vendor Independence, Application Independence, Feasibility, Uniqueness from other approaches, Availability of Information, and Testability within a two week period to determine viability for use as a verification approach with the following results.

> Two approaches, the User Oriented Microprocessor ATP and the Adam ATP, were eliminated due to lack of information. However, indications are that they are similar to the AN/AYK-15A ATP.
>
> Although the Functional approaches yield similar verification approaches, none of the methods were judged to be significantly similar to another to justify failing them since each of the functional methods meet different design requirements.
>
> There were some application and vendor dependencies in some of the methods. However, these dependencies were considered to be correctable.
>
> Two approaches, the ISP method and the Analytic method, were eliminated due to feasibility. These were eliminated because the tools required to implement the method were not sufficiently developed for use at this time or in the near future.

Table 6-2 further summarizes the pass/fail analysis presented in this section.

Table 6-2. Pass/Fail Analysis

Method	Pass/Fail	Failure Criteria	Generic Type
Adam ATP	Fail	Not Unique*	Functional
AN/AYK-15A ATP	Pass		Functional
Random Instructions	Pass		Random Instruction
Analytical Approach	Fail	Not Feasible	Analytical
AVP	Pass		Functional
Diagnostic	Pass		Functional
FTP	Pass		Functional
ISP	Fail	Not Feasible	Functional
Lockstep	Pass		Lockstep
User Oriented Micro Computer (ATP)	Fail	Not Unique*	Functional

* Subset of AN/AYK-15A ATP

THIS PAGE INTENTICNALLY LEFT BLANK

7.0 ANALYSIS

The previous section provided detailed descriptions of the different verification approaches and applied the pass/fail criteria to them. This section provides, for those verification approaches that remain, detailed cost information and the quality data, as required by the cost model. These data are analyzed and the results interpreted.

This section consists of four parts. The first discusses the various test configurations. The second presents cost data for the different verification approaches. Next, the quality data is presented and analyzed. Finally, conclusions are drawn based on the first three sections.

7.1 TEST CONFIGURATION

Each verification approach may be implemented utilizing a variety of test configurations. For reasons of practicality the following generic test configurations have been identified as being representative of all possible test configurations in degrees of complexity:

Manual	- minimal hardware configuration
Automatic	- simple I/O, self-hosted control
Master/Slave	- auxiliary processor to support testing.

The Manual Test Configuration will be discussed first, followed by the Master/Slave Test Configuration. The Automatic Test Configuration will be described last, with arguments presented for considering it as a subset of the Master/Slave Test Configuration. Each test configuration is further broken down into hardware and software components. Vendor supplied and Air Force supplied items are separately identified. Cost items are also identified as applying to either the vendor or to the Air Force.

7.1.1 Manual Test Configuration

7.1.1.1 Description

The Manual Test Configuration is, by definition, the minimal hardware configuration necessary to perform adequate self-documenting verification to the MIL-STD-1750 architecture. The test configuration requires all hardware to be supplied by the vendor, and makes no equipment requirements of the Air Force other than the power to run the test equipment and computer. The hardware necessary for the manual configuration is as follows:

- MIL-STD-1750 Computer
- Ground Support Equipment including the following:
  - Memory Load Facility
  - Hardcopy Device (Printer or Typewriter Terminal)
  - Memory and Register Display/Modify Facility
  - Program Start and Stop/On Compare Facility

Figure 7-1 depicts a typical Manual Testing Configuration. Software is supplied by the Air Force and the vendor. The architecture verification program for the MIL-STD-1750 is developed by the Air Force, assuming a standard subroutine linkage for output messages to the vendor supplied hardcopy device. The vendor must develop the output subroutine.

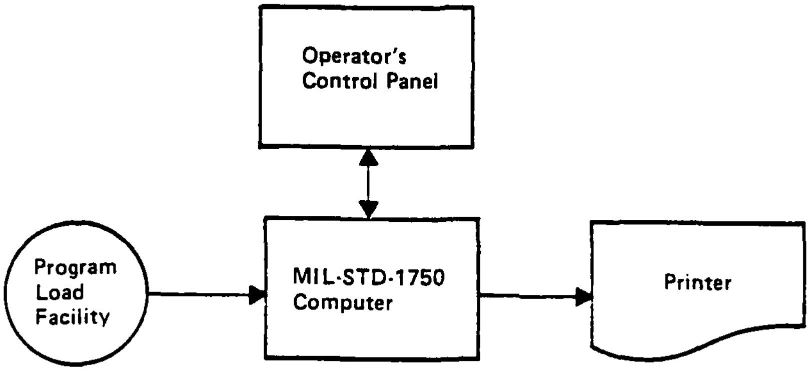


Figure 7-1. Manual Test Configuration

The certification scenario would then be as follows:

1. The Air Force gives a copy of the verification program source code to the vendor and an output subroutine specification.

2. The vendor develops the subroutine to handle hardcopy output, and then prepares the load tapes by assembling source and link editing with the output subroutine.

3. The vendor brings the computer, test equipment and program load tapes (containing load modules of the verification program) to SEAFAC.

4. Under SEAFAC observation the vendor first clears main storage to all zeros (which the Air Force verifies), and then loads and executes the verification program which prints out the results.

5. Various random memory locations are inspected to verify the integrity of the verification program on the load tape.

6. The vendor leaves the load tapes and assembler/link editor listings with SEAFAC for archive purposes.

7.1.1.2 Analysis

The Manual Test Configuration offers a low cost testing environment for the Air Force to conduct the MIL-STD-1750 certification process. The hardcopy device provides automatic self-documentation of test cases under control of the verification program. The vendor has pre-verification offsite testing ability. After successfully testing offsite, onsite compliance to the verification procedure should be simpler. The risk of the vendor fraudulently modifying the verification program to compensate for architectural inadequacies is offset by the Air Force's keeping the load tapes for future reference if necessary. The disadvantages associated with this test configuration center around its limitations associated with manual intervention. If the verification program requires several system loads, then the memory loading procedure could become a critical factor. If several loads could not be made from the same IPL tape, several tape mounts are necessary. This data will be described further in the analysis of subsequent proposed verification approaches.

7.1.1.3 Costs

Air Force costs associated with the Manual Testing Configuration are limited to software development and maintenance costs and normal certification personnel staffing hours as required by the selected verification program and testing procedure. Hardware costs to the Air Force are non-existent since all hardware is supplied by the vendor.

Vendor costs include hardware costs, software costs, and computer operator cost. The hardware necessary for the manual test configuration can be considered to be the standard equipment used for normal computer development with the exception of the hardcopy device. Software cost would include the development of the output subroutine and the generation of the load tapes. Personnel costs would simply cover the vendor representative presence to mount the load tapes and start the verification program running.

7.1.2 Master/Slave Test Configuration

7.1.2.1 Description

The Master/Slave Test Configuration is the most complex hardware configuration considered to perform the verification to the MIL-STD-1750 architecture. The test configuration requires both vendor and Air Force supplied hardware and software. The hardware associated with the Master/Slave Test Configuration is as follows:

ITEM	SUPPLIER
Master Computer with the following: - Hardcopy Device - Auxiliary Storage - Console/Terminal - Communications Link - MIL-STD-1553 - RS-232	Air Force
MIL-STD-1750 Computer	Vendor
I/O Channel on MIL-STD-1750 or Channel Adapter (i.e., 1553, RS232, etc.)	Vendor
Ground Support Equipment with - Main Storage Load Facility - Main Storage and Register Display/Modify Facility - Program Start and Stop on Compare Facility	Vendor

The software necessary to carry out the certification under the Master/Slave Test Configuration would be as follows:

Verification Program for MIL-STD-1750	Air Force
Bootstrap Program on MIL-STD-1750	Air Force
Control Program on Master	Air Force
I/O Interface Test Programs	Air Force
Utility Programs	(Standard on Master Computer)

Inrut/Output Subroutines Vendor

Figure 7-2 depicts the Master/Slave Test Configuration. The certification scenario would be as follows:

1. The Air Force gives a copy of the boctstrap program source code and I/O Test Programs (both require vendor supplied I/O subroutines) to the vendor and specifies I/O subroutine requirements.

2. The vendor develops subroutines to handle I/C, and then prepares a bcotstrap load tape by assembling bootstrap program source and link editing it with the I/O subroutines. An I/O Test Program load tape is similarly developed.

3. The vendor brings computer test equipment and load tapes to SEAFAC.

4. Both the vendor and the Air Force connect the Master computer to the vendor's I/O channel.

5. The I/O Interface Test Programs are loaded and run to verify the communication interface between the unit under test and the Master computer.

6. Under SEAFAC observation the vendor loads the bootstrap and I/O programs and starts bootstrap execution.

7. SEAFAC personnel start the control program on master computer.

8. The verification program is transmitted to the vendor's MIL-STD-1750 computer by the Master computer and control is given to the verification program. When the verification program finds an error or requests more test cases, the Master computer is notified over the communication link. Test results are documented on some form of hardcopy device (typewriter terminal or printer) associated with the Master computer system.

7.1.2.2 Analysis

The Master/Slave Test Configuration offers the greatest degree of automation available for the Air Force to conduct the MIL-STD-1750 certification process. Furthermore, this approach makes certain verification approaches feasible (Random, AVP, Lockstep) that would not have been feasible due to implementation restrictions discussed in subsequent sections of this document. The Master/Slave Test Configuration also makes use of the available peripherals associated with the Master computer and facilitates excellent tracking capability

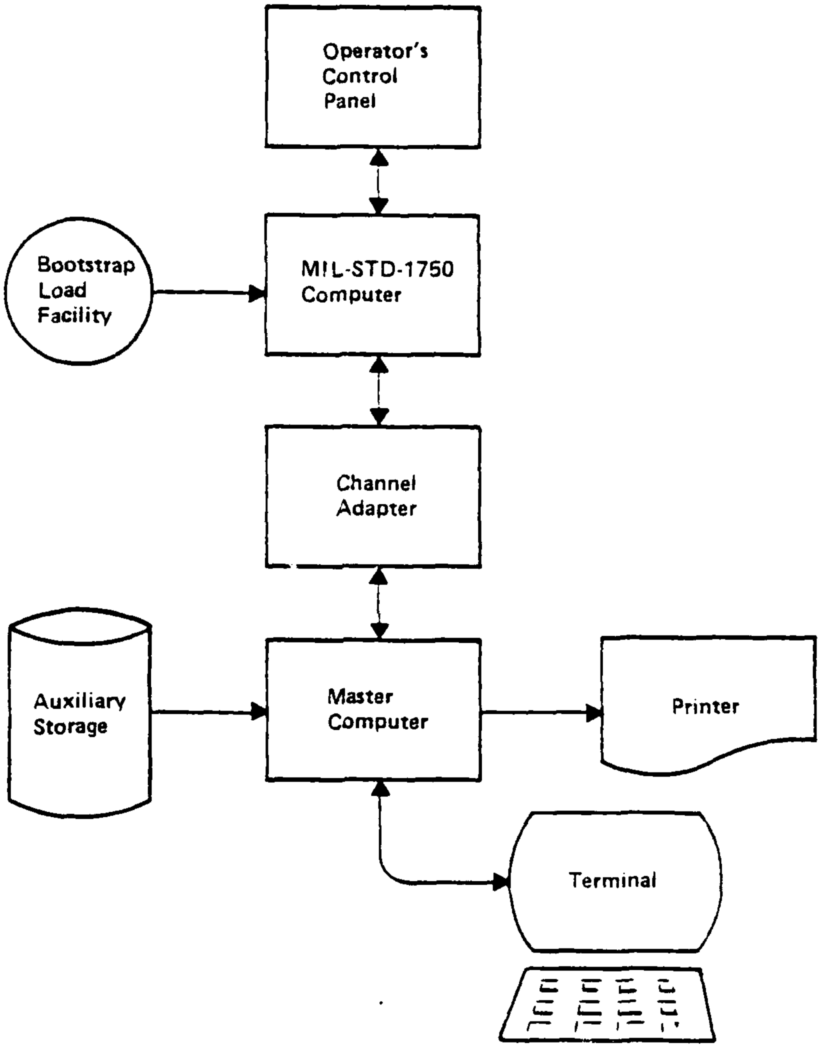


Figure 7-2. Master/Slave Test Configuration
(Sheet 1 of 1)

(ex. spooling intermediate results to mass storage to be printed at a later time). The Master/Slave Test Configuration brings into play, as well, the powerful computation power of the Master and associated support software.

The disadvantages associated with this test configuration are cost and complexity. The cost of the Master computer is considerable under most circumstances, but for purposes of this study, the cost of the Master computer, its peripherals and the MIL-STD-1553 or RS-232 communication link are to be considered as zero since they are existing SEAFAC assets. From the vendor's perspective, the impact of the Master/Slave Test Configuration (besides the additional input/output routines and I/O channel required) is the potential limitation concerning pretest. Depending on the verification approach selected to run under this test configuration, the vendor will be able to utilize parts of the test code made available prior to in-house testing by the Air Force at SEAFAC. In the area of complexity, the Master/Slave Test Configuration puts an additional cost burden on the Air Force for developing the bootstrap program for the MIL-STD-1750 and control program for the Master computer, as well as the I/O Interface Test Programs.

7.1.2.3 Costs

Air Force's cost associated with the Master/Slave Test Configuration can be segmented into two areas; (1) hardware costs and (2) software costs. The hardware costs to the Air Force would include the cost of the Master computer with related peripherals and communications link which have already been established to be zero. The software costs are compounded by the additional development costs of the bootstrap program for the unit under test and the control program for the Master computer along with the normal cost of writing (and maintaining) the selected verification program. The I/O Interface test programs are necessary to integrate the vendor and the Air Force hardware prior to actual verification.

SEAFAC personnel requirements to monitor the execution of the verification program would be minimal since the approach is fully automated. A technician familiar with the Air Force's I/O Interface should be available to assist the vendor when the vendor connects to that I/O Interface. Vendor costs consists of the normal computer and Ground Support Equipment augmented by the I/O channel requirement. Software provided by the vendor consist of the I/O subroutines and the bootstrap and I/O Test Program load tapes.

AD-A099 260 IBM FEDERAL SYSTEMS DIV OWEGO N Y F/G 9/2
MIL-STD-1750 CERTIFICATION STUDY.(U)
FEB 80 M L KUSHNER, D C REISIGER, W J TRACZ F33657-79-M-0858
UNCLASSIFIED IBM-6176175A NL

2 OF 3
AD
A099260

7.1.3 Automatic Test Configuration

7.1.3.1 Description

The Automatic Test Configuration provides a workable standalone self-documenting system, with reasonable cost and performance to conduct the certification of the MIL-STD-1750 architecture. This test configuration places the same software and hardware requirements on the vendor as described in the Master/Slave Test Configuration. The hardware associated with the automatic test configuration is as follows:

ITEM	SUPPLIER
Certification Peripherals	Air Force
- Hardcopy Device	Air Force
- Auxiliary Storage (Tape/Floppy Disk)	Air Force
- I/O Adapter (MIL-STD-1553/RS-232)	Air Force
MIL-STD-1750 Computer	Vendor
I/O Channel or I/O Adapter on MIL-STD-1750 (MIL-STD-1553/RS-232)	Vendor
Ground Support Equipment	Vendor
- Main Storage Load Facility	
- Main Storage and Register Display/Modify Facility	
- Program Start and Stop on Compare Facility	

The software necessary to carry out the verification under the Automatic Test Configuration would be as follows:

Verification Program for MIL-STD-1750	Air Force
Bootstrap Program on MIL-STD-1750	Air Force
Control Program to Access Auxiliary Storage	Air Force
I/O Interface Test Programs	Air Force
Input/Output Subroutines	Vendor
Utility Programs	(Standard on Development Computer)

Figure 7-3 depicts the Automatic Test Configuration.

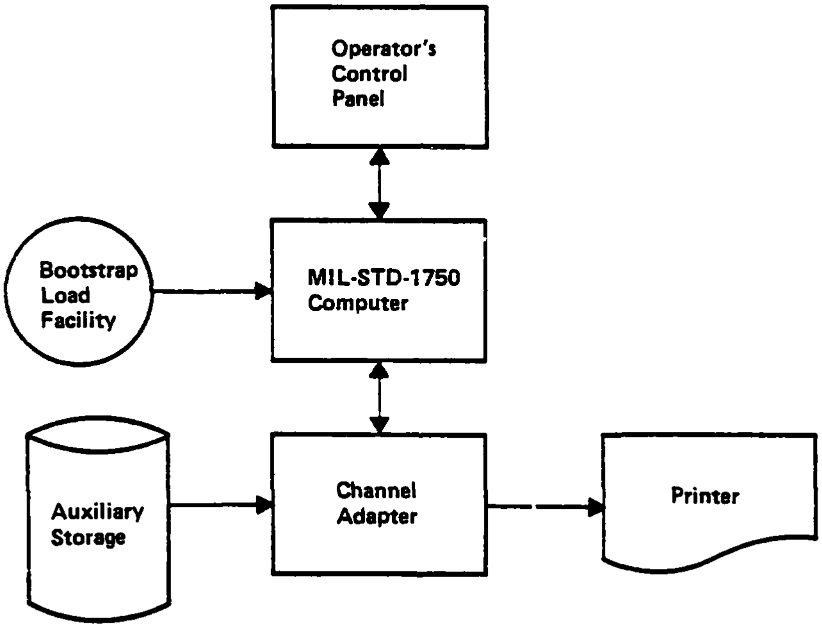


Figure 7-3. Automatic Test Configuration

The certification scenario would be as follows:

1. The Air Force gives a copy of the bootstrap program source code and I/O Test Program (both require vendor supplied I/O subroutines) to the vendor and specifies I/O subroutine requirements.

2. The vendor developes subroutines to handle I/O, and then prepares a bootstrap load tape by assembling the bootstrap program source and link editing it with I/O subroutines. An

I/O Test Program load tape is similarly develcped.

3. The vendor brings his computer, test equipment and load tape with the bootstrap program and I/O subroutines to SEAFAC.

4. Both the vendor and the Air Force connect SEAFAC's certification peripherals to the vendor's I/C channel.

5. I/O Interface test programs are loaded and run to verify the communication interface between the unit under test and the Air Force supplied peripherals.

6. Under SEAFAC observation, the vendor lcads the bcotstrap and I/O programs, and starts their execution.

7. The bootstrap program loads the verificaticn program from the auxiliary storage and commences execution. Errors or other messages will be logged out on the printer. Subsequent program loads from auxiliary storage will be made under program contrcl. No manual interventicn is necessary.

7.1.3.2 Analysis

The Automatic Test Configuration offers the simplest fully automated, self-documenting system for the Air Force to conduct the MIL-STD-1750 certification process. This approach facilitates certain verification approaches, but limits the potential dynamic nature of certain test procedures (Random) which will be explained in subsequent sections. The main disadvantage of this approach is that it requires the Air Force to purchase, integrate and maintain the selected peripheral devices. Vendor requirements are the same as fcr the Master/Slave Test Configuration, but the implications of pre-verification occurring cffsite using the Air Force develcped verificaticn program becomes more feasible since the Automatic Test Configuration is easier for the vendor to implement than the Master/Slave Test Configuration. The Automatic Test Configuration might alsc be considered portable if the Air Force implemented it as such.

In summary the Automatic Test Configuration will give the Air Force considerably less function than the Master/Slave Test Configuration, at the same time costing more because cf the purchase of additional peripheral devices. The vendor requirements are the same and the Air Force's software costs are the same as for the Master/Slave Test Configuration, thus indicating that the Master/Slave Test Configuration offers a superior type of approach.

7.1.3.3 Costs

The cost to the Air Force to implement the Automatic Test Configuration to support the certification of the MIL-STD-1750 computer consists of the following items:

- Purchase of Certification Peripherals (Auxiliary Storage and Printer)
- Development of Interface
- Development of Verification Program
- Development of Bootstrap Program
- Development of the I/O Interface Test Programs
- Maintenance of all Software

Vendor costs would be the same as with the Master/Slave approach. SEAFAC personnel requirements, to monitor the execution of the certification program, would be minimal since the approach is fully automated. A dynamic approach could be selected requiring the substitution of auxiliary storage to handle additional data requirements (floppy disks or tapes), thus reducing the system to a semi-automated state. The I/O Interface bring-up would require a technician familiar with the Air Force's I/C Interface to assist the vendor.

7.1.4 Comparison

All three approaches described in the preceding sections provide the Air Force with some form of certification facility. The Manual Test Configuration offers the least amount of automation while the Master/Slave Test Configuration has the most. The Manual Test Configuration restricts the implementation of the verification program to certain verification approaches, while the Master/Slave Test Configuration places no limitations on the verification approach selected. The Manual Test Configuration provides convenient offsite pre-testing by the vendor, while the Master/Slave Test Configuration has some potential implementation dependent limitations.

The Automatic Test Configuration costs more for the Air Force to implement because of the additional hardware purchased, while providing less function than the Master/Slave Test Configuration. For this reason, the Automatic Test Configuration will be dropped from further consideration as a feasible test configuration. Subsequent verification approach analysis will center around the remaining two. Table 7-1 summarizes the cost breakdowns and components associated with each test configuration.

Table 7-1. Test Elements

	Master/Slave	Automatic	Manual
Vendor Costs			
Software	I/O Subroutines Load Tape Support Software	I/O Subroutines Load Tape Support Software	Output Subroutine Load Tape Support Software
Personnel	GSE Operator	GSE Operator	GSE Operator
Hardware	1750 Computer Special I/O Interface	1750 Computer Special I/O Interface	1750 Computer None
	Master Computer (For Verification Pre-Test)	Peripherals (For Verification Pre-Test)	Printer

Table 7-1. Test Elements (cont)

	Master/Slave	Automatic	Manual
Air Force Costs			
Software	Verification Program	Verification Program	Verification Program
	Bootstrap Source Control Program	Bootstrap Source Utility Program - Offload Test Code	None Utility Program - Offload Source
	Support Software I/O Test	Support Software I/O Test	Support Software I/O Test
Hardware	Interface	Interface	None
	Master Computer/ Peripherals	Peripherals	None
Personnel	Development Programmers	Development Programmers	Development Programmers
	Maintenance Programmers	Maintenance Programmers	Maintenance Programmers
	Test Operator/ Observer	Test Operator/ Observer	Test Operator/ Observer
	Integration Technician	Integration Technician	None
Other	Test Procedure	Test Procedure	Test Procedure
	Utilize Master Computer for Test	Generate Test Tape	None

7.2 TEST APPROACHES

In this section, the test approaches remaining after applying the pass/fail criteria will be discussed under two test configurations, Manual and Master/Slave. Each approach will be analyzed and described in perspective with "being implemented" as a verification program for the MIL-STD-1750. Variations of certain approaches will also be covered. The proposed approaches discussed in the following sections are:

- AN/AYK-15A Acceptance Test Program
- Random Instructions
- Architectural Verification Program (AVP) - System 360/370
- Diagnostic
- Functional Test Program (FTP)
  - with existing FTP available
  - with no FTP available
- Lockstep
  - with Trace feature
  - without Trace feature
  - with Simulator instead of "golden" computer
  - with predetermined results

In order to facilitate the comparison of comparable quantities, the following assumptions are made concerning each projected approach.

1. Each (non-Random) verification program has on the average 25 test cases per instruction resulting in approximately 5,000 separate test cases being generated for an architecture of the MIL-STD-1750 class.

2. A programmer productivity of 2 test cases per day is assumed. The Air Force's suggested rate for Operational Flight Program development is 110 lines of High Order Language instruction statements per month, and 75 lines of machine language instruction statements per month. They are modified for calculating the development cost of verification programs based on the following reasons:

   a. Verification programs are greatly simpler in complexity and organization than operational flight programs.

b. Verification programs are easily modularized into different program segments.

c. There is a minimal amount of inter-module communication in verification programs.

d. Each module in a verification program contains only very simple program logic.

e. Each module's function is extremely repetitive in nature (i.e., load data values, perform operation, and check result with the expected result).

f. The main storage area (reserved for constants and expected results) comprises 30 to 50 percent of each test module.

Therefore, a programmer productivity rate of 200 lines of High Order Language instruction statements per month, 180 lines of machine language instruction statements for control program code, and 280 lines of machine language instruction statements for test program code per month are used in calculating the development cost for the verification programs.

3. Software maintenance costs are projected assuming that two errors are found per a thousand lines of delivered code and that each error takes 1 man week to correct. Software maintenance costs are based on a ten year life of the verification program.

4. Total recurring costs are based on a ten year life of the program and 30 computers being certified.

5. The MIL-STD-1750 computer has a 32K, 16-bit words of main storage.

6. The VAX 11/780 computer system, the MIL-STD-1553 I/O channel, and the RS-232 I/O channel are available to support the Master/Slave approach at zero cost.

7. The time required to perform validation does not include the time allocated to cabling up the computer and verifying the I/O interface. It is assumed that a 8-hour time slice would be more than adequate to support this activity.

8. For calculation purposes, the unit under test computer is assumed to be a 500 KOP machine, and the Master computer is capable of executing one million instructions per second and prints at a 300 line per minute rate.

9. A cost figure of $70K per man year is used in developing dollar cost totals.

NOTE: Contrary to intuition, the type of program known as a "Control Program" varies both in size and function throughout the 12 approaches analyzed. Therefore, cost figures for each approach will reflect this variance between control programs. Similarly, the overhead processing per test case for each approach varies from 50 instructions for AN/AYK-15A, Diagnostic and FTP, to 1,000 for Lockstep, 10,000 for AVP and 15,000 for Random. These numbers were developed from the data gathered during the first phase of the study and represent the type of processing required in the test case initialization and execution followed by the verification of results. The first three approaches invoke in-line tests, thus taking little overhead. The remaining approaches require operating system overhead, or control card processing (AVP), or simulation and incur a large amount of additional processing.

7.2.1 AN/AYK-15A ATP in a Manual Test Configuration

7.2.1.1 Description

A verification program developed from the AN/AYK-15A ATP, targeted for a Manual Test Configuration, would yield a satisfactory and thorough static test of the MIL-STD-1750 instruction set. Using the existing AN/AYK-15A ATP as a starting point, the following modifications must be made. Each test module must be analyzed for content with irrelevant test cases excluded and relevant test cases added to increase coverage. The supervisor program must also be modified to communicate with the newly defined I/O interface associated with the manual test configuration. The estimated size of the finished verification program is 96K, requiring three program loads.

7.2.1.2 Non-Recurring Start-Up Costs

The cost of implementing a verification program based on modifying the AN/AYK-15A ATP under a Manual Test Configuration would consist of the following software development components.

Modification of AN/AYK-15A ATP (Modify 30% of 30K BAL* = 9K) (Write 1K BAL Control Program)	=	3.1 man years
Source Tape Generator Program (500 BAL)	=	0.3 man years
Test Plan Document	=	0.3 man years
	TOTAL	3.7 man years

```
+-------------------------------------+
|*BAL means Basic Assembler language|
|           or, equivalently,         |
| Machine Language Instructions       |
| HOL means High Order Language       |
|                                     |
|Note that where HOL and BAL figures|
|appear on the same line, this means|
|that portions of the program will  |
|be written in each language.       |
+-------------------------------------+
```

7.2.1.3 Recurring Costs

The recurring costs associated with a verification approach based on the modification of the AN/AYK-15A ATP under a Manual Test Configuration consist of two major components - verification cost, and the cost to sustain the software. The recurring costs reflects the cost per computer tested, assuming 30 computers tested over a 10 year period. The verification costs are proportional to the staffing allocated during each vendor verification and the actual time it takes to complete each verification procedure. The cost to sustain the software based on the number of source lines of code (not program size) are calculated in the following table. An additional cost, although nominal in nature, is the cost of generating the verification program source tape for the vendor to make IPL tapes from. This cost is considered part of the verification cost. Table 7-2 contains a summary of recurring costs.

7.2.1.4 Time Required to Perform Validation

The time required to perform validation using a verification program based on the modification of the AN/AYK-15A ATP under a Manual Test Configuration consists of the summation of mount times, memory load times and execution times. Assuming 5 minutes to mount the tape, 3 minutes to load memory and 8 minutes to process the test modules on each load tape and print out the results, the maximum time to run the verification test error free would be calculated as follows:

Verification time = N_1 * 5 minutes to mount each tape +
N_2 * 3 minutes to load and go +
N_2 * 2 minutes to execute the program and print out results

where:

N_1 = number of tape mounts = 3 worst case

N_2 = number of program loads = 3

Verification time = $3 * 5 + 3 * 3 + 3 * 8$
= 48 minutes

7.2.1.5 Impact to SEAFAC Resources

The implementation of a verification approach based on the modification of the AN/AYK-15A ATP under a Manual Test Configuration would require the use of MIL-STD-1750 support software (cross assembler, linking loader, and simulator) on the development computer system. SEAFAC personnel would develop and maintain the certification program as well as prepare the source tape to give to each vendor. During the verification process, there would be no impact on the development computer, but SEAFAC personnel would be required to assist in the integration, initiation, and observation of the verification program executing on the unit under test.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-2.

Table 7-2. Cost Summary for the AN/AYK-15A ATP Approach Under A Manual Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Computer	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Modification of AN/AYK-15A ATP	3.1	217
- Source Tape Generation Program	0.3	21
Other		
- Test Plan Document	0.3	21
TOTAL	3.7	259
Recurring Costs/Computer		
Hardware		
- Maintenance		0
Software		
- Maintenance 40K * 2 errors/K * $1,400/30	0.052	3.7
Personnel		
- Coverage to Observe Execution and Analyze Results (2 People for 1 Week)	0.04	2.8
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.096	6.78

7.2.2 AN/AYK-15A ATP in a Master/Slave Test Configuration

7.2.2.1 Description

A verification program based on modifying the AN/AYK-15A ATP, targeted for a Master/Slave Test Configuration would result in a verification process closely resembling one based on the FTP or Diagnostic approach. The AN/AYK-15A test modules would be analyzed for content with the irrelevant test cases deleted and additional test cases added to increase coverage. The supervisor program would be modified to communicate with the control program on the Master to facilitate the loading of test modules into the unit under test and the passing of test results back to the Master for recording. The control program on the Master would handle all I/O and would be initially invoked by a bootstrap program loaded on the unit under test (slave) by the vendor at the start of the verification process. The approximate size of each of the software modules follows:

Supervisor	=	8K words
Test Modules	=	88K words
Control Program	=	8K words
Bootstrap Program	=	1K words

7.2.2.2 Non-Recurring Start-Up Costs

The cost of implementing a verification program based on modifying the AN/AYK-15A ATP under a Master/Slave Test Configuration would consist of the following software development components:

Modification of AN/AYK-15A ATP (Modify 30% of 30K BAL = 9K) (Write 1K BAL Supervisor Program)	=	3.1 man years
Control Program (Master) (2K HOL; 500 BAL)	=	1.1 man year
Bootstrap Program (400 BAL)	=	0.3 man years
Source Tape Generator Program (500 BAL)	=	0.3 man years
I/O Test Programs (1K BAL)	=	0.5 man years
Test Plan Document	=	0.3 man years
TOTAL		5.6 man years

7.2.2.3 Recurring Costs

The recurring costs associated with modifying the AN/AYK-15A ATP to run under a Master/Slave Test Configuration consists of the usage of the Master computer (zero cost) and the staffing for the integration, initiation and observation of the verification process, plus the analysis of the results. The second major cost is the cost to sustain all software. Table 7-3 contains a summary of recurring costs.

7.2.2.4 Time Required to Perform Validation

The time required to perform the complete verification process is calculated as follows:

Verification time = Bootstrap Load and Go (5 minutes) +

Master Computer Control Program Initialization (5 minutes) +

Verification Program Execution Time (1 minute) +

I/O Transfer Time (3 seconds)

= 11 minutes.

Table 7-3. Cost Summary for the AN/AYK-15A ATP Approach Under A Master/Slave Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.1	77
- Modification of AN/AYK-15A	3.1	217
Other	0.5	35
- Test Plan Document	0.3	21
TOTAL	5.6	392
Recurring Costs/Computer		
Hardware		
- Maintenance		0
Software		
- Maintenance 44.4K * 2 errors/K * $1,400/30	0.059	4.14
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.107	7.5

7.2.2.5 Impact to SEAFAC Resources

The implementation of a verification program by modifying the existing AN/AYK-15A ATP to run under a Master/Slave Test Configuration would utilize the support software (MIL-STD-1750 cross assemblers, linking loader, and simulator) as well as normal system utilities on the Master computer during system development. During the verification process, the Master computer would play a passive role serving as an I/O controller for the verification program. SEAFAC personnel would be required to develop and sustain the verification program, as well as the bootstrap and control programs. During testing, SEAFAC personnel must also supervise the integration and initialization of the verification process.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-3.

7.2.3 Random Instruction in a Manual Test Configuration

7.2.3.1 Description

A verification program developed using a Random Instruction design philosophy targeted for a Manual Test Configuration would result in a verification process with certain severe limitations given the current memory constraints of the computer. The Random tests approach in a standalone mode consist of a random instruction sequence generator, a supervisor program and a high fidelity/quality "golden" simulator. The random instruction sequence generator generates a sequence of instructions that are executed by the hardware and simulated by the simulator The supervisor program then compares the generated and simulated results and prints out the results.

The approximate size of each software component is:

Control Program	=	4K words
Random Instruction Generator	=	24K words
"Golden" Simulator	=	300K words

Given the 32K memory size specification for all units under test, a dynamic random instruction verification program under a Manual Test Configuration would not be possible to implement. An alternative is a static approach. Random sequence of instructions are generated offline and saved. These instruction sequences are then simulated and their results saved. Load tapes consisting of a supervisor program, sets of random instruction sequences and test results are generated.

The load tapes are executed on the unit under test. The supervisor program sequentially executes the random instruction test sets, comparing generated values to expected values and printing the results. The number of random instruction sequences executed is limited by the time allocated for the validation process, and the time it takes to load and execute the test cases.

If the supervisor program takes 8K words, that leaves 24K words for test cases, expected results, and generated results. Each test set is (on the average) 40 words long, the expected results for each test set is 60 words; each test set effectively contains 10 executable test cases. In the 24K of data space, 240 test sets could be allocated. Therefore to facilitate the execution of 125,000 test sets with a total of 1,250,000 test cases being executed, 521 separate memory loads would have to be made. Furthermore, the Air Force would have to supply the vendor with 12.75 megawords worth of data to generate the load tapes. ((521 loads * 24K words) + 8K.)

It has previously been stated that when the Random Instruction method has been applied to verify the architecture of a computer which is capable of executing four million instructions per second, that most, if not all, of the architectural discrepancies are discovered after one hour of processing. In order to obtain the necessary coverage, confidence, and completeness, this interprets to the execution of 125,000 sets of random instruction sequences with each having an average length of 10 executed instructions (although the sequences are 32 instructions in length, the percentage distribution of branch instructions randomly appearing brings the average number of instructions executed per test set sequence to 10). Therefore, 1,250,000 random instructions are tested per hour (on the four million instructions per second computer), with each instruction in itself being a test case. The term "test case", therefore, when used in conjunction with the random instruction approach, refers to a single instruction which is generated, and verified.

NOTE: The execution of 1,250,000 randomly generated instruction test cases has been judged a priori to be comparable from a quality viewpoint to the 5,000 manually generated test cases used in other approaches.

7.2.3.2 Non-Recurring Start-Up Costs

The cost for implementing a verification program based on the Random Instruction approach under a manual test configuration would consist of the following software development costs:

Random Instruction Generator (6K HOL; 1K BAL)	=	3.0 man years
Simulator (12K HOL; 2K BAL)	=	6.0 man years
Supervisor Program (4K BAL)	=	1.8 man years
Tape Generator Program (500 BAL)	=	0.3 man years
Test Plan Document	=	0.3 man years
	TOTAL	11.4 man years

7.2.3.3 Recurring Costs

The recurring costs associated with the Random Instruction approach implemented under a Manual Test Configuration consists of three significant components. First, the cost of generating 1,250,000 test cases and results on the development system and creating the test case source tapes for the vendor, prior to in-house testing. Second is the personnel requirements necessary to observe the manual test procedure taking place. The time required for generating the 1,250,000 test cases and results based on 15,000 instructions being executed to generate the data necessary for each test case on a computer (capable of executing one million instructions per second) is:

$$15{,}000 * 1{,}250{,}000 \text{ instructions} \bullet \frac{1 \text{ sec}}{1{,}000{,}000} \text{ instructions} = 18{,}750 \text{ sec}$$

Therefore it would take 5 hours, 12 minutes of CPU time to generate the data to be put on a single tape plus the I/O transfer time for 12.75 megawords. The cost of sustaining the certification program is the third component of recurring costs.

7.2.3.4 Time Required to Perform Validation

Using the formula developed previously, the verification time to load, execute and print out the results of 1,250,000 randomly generated test cases would be as follows:

Verification time = N_1 * 5 minutes to mount each tape +
N_2 * 3 minutes to load and go +
N_2 * 2 minutes to execute the program and print out results

where:

N_1 = number of tape mounts = 521 worst case

N_2 = number of program loads = 521

Verification time = 521 * 5 + 521 * 3 + 521 * 2
= 5210 min = 86.8 hours = 11 days (8 hours/day)

7.2.3.5 Impact to SEAFAC Resources

Using a Manual Test Configuration under which to implement a Random Instruction test approach places a significant burden on SEAFAC personnel to be present during the 11 days of testing. Other requirements would be the normal support software (cross assembler, linking loader and simulation); plus the development of the random instruction generation program. As previously mentioned, the generation of the random instructions, though completely automated requires a significant amount of CPU time on the development computer.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-4.

Table 7-4. Cost Summary for the Random Instruction Approach Under A Manual Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Computer	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- "Golden" Simulator	6.0	420
- Random Instruction Generation Program	3.0	210
- Source Tape Generator Program	0.3	121
- Supervisor Program	1.8	126
Other		
- Test Plan Document	0.3	21
TOTAL	11.4	799
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 25.5K * 2 errors/K * $1,400/30	0.034	2.38
Personnel		
- Coverage to Observe Execution and Analyze Results (1 Person for 2 Weeks)	0.04	2.8
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.078	5.46

7.2.4 Random Instruction in a Master/Slave Test Configuration

7.2.4.1 Description

The Random Instruction design approach is ideally suited to the Master/Slave Test Configuration for implementing the MIL-STD-1750 verification program. Under this approach, the random instruction generator and "golden" simulator would reside on the Master computer. The unit under test (slave) would contain supervisor programs whose functions are to request test cases, execute them, and send back the results to a control program running in the Master computer. The control program supervises the generation, simulation and comparison functions on the Master computer and logs the results. The Air Force would send the vendor a copy of the bootstrap program, and the vendor would bring it in IPL format along with the I/O routines at the time of certification. The size of each program module is described in the previous section.

7.2.4.2 Non-Recurring Start-Up Costs

The costs of implementing a verification program based on the Random Instruction approach under a Master/Slave Test Configuration would consist of the following software development costs:

Random Instruction Generator (Master) (6K HOL; 1K BAL)	=	3.0 man years
"Golden" Simulator (Master) (12K HOL; 1K BAL)	=	6.0 man years
Control Program (Master) (3K HOL; 1K BAL)	=	1.8 man years
Supervisor Program (Slave) (3K BAL)	=	1.4 man years
Bootstrap Program (400 BAL)	=	0.3 man years
Source Tape Generator Program (500 BAL)	=	0.3 man years
I/O Test Programs (1K BAL)	=	0.5 man years
Test Plan Document	=	0.3 man years
	TOTAL	13.6 man years

7.2.4.3 Recurring Costs

The recurring costs associated with the Random Instruction approach under the Master/Slave Test Configuration consist of computer usage on the Master computer (zero costs) and staffing for whatever portion of the test is required, especially the analysis of the results. This cost is augmented by system integration and initiation costs. The final cost component is the cost of sustaining the software.

7.2.4.4 Time Required to Perform Validation

The time required to perform the complete verification test consists of the summation of Bootstrap Load and Go time, Master Computer Control Program Initiation time, Test Case Generation and Execution time, and I/O Transfer time. An estimate of these times follows:

Test Case Generation, Simulation and Verification
Master
15,000 instructions per test case overhead to generate, simulate, and verify results *
1,250,000 test cases *
1 sec/1,000,000 instructions (speed of master)

Total time of Generation/Verification = 18,750 seconds = 313 min

Test Case Execution on Slave
1,000 instructions per test case overhead *
1,250,000 test cases *
1 sec/500,000 instructions (speed of slave)
= 2,500 sec = 42 min

Data Transfer
125,000 test set * 40 words of instructions per test set +
125,000 test set * 60 words of results per test set
= 12.5 megawords / 30K words/sec = 417 seconds = 7 min

Verification time = Bootstrap Load and Go (5 minutes) +
Master Computer Control Program Initialization (5 minutes) +
Test Case Generation Time (313 minutes) +
Test Case Execution Time (42 minutes) +
I/O Transfer Time (7 minutes)
= 6 hours, 12 minutes.

NOTE: The major component in the verification time is the time to generate, simulate and verify the random instructions. This component is in turn proportional to the overhead (15,000 instructions) taken to perform this task. This figure assumes a 10,000 to 1 performance ratio on the simulator, which is a figure characteristic of simulators developed in a high order language. The total verification time is a worst case analysis and makes no assumptions regarding potential speed up if overlapped processing is implemented. Total time is improved only two percent if a parallel I/O channel is used instead of MIL-STD-1553 channel, and would be degrated by 17 percent if a RS-232 channel is used.

7.2.4.5 Impact to SEAFAC Resources

A verification program based on the Random Instruction design philosophy targeted for Master/Slave Test Configuration would utilize the Master computer both during the development phase and during the verification procedure. SEAFAC personnel would be required to write all software modules for the Master and Slave. This would include the use of such normal support software available on the Master (compiler, editor, linking loader) as well as MIL-STD-1750 support software, (cross assembler, linking loader, and simulator). During the actual verification process, a certain portion of the Master computer's computational power must be dedicated to the generation, simulation, verification and documentation of the test cases.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-5.

Table 7-5. Cost Summary for the Random Instruction Approach Under A Master/Slave Test Configuration

	Cost	
Item	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.8	126
- "Golden" Simulator	6.0	420
- Supervisor Program on Slave	1.4	98
- Random Instruction Generation Program	3.0	210
- I/O Test Programs	0.5	35
Other		
- Test Plan Document	0.3	21
TOTAL	13.6	952
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 28.9K * 2 errors/K * $1,400/30	0.039	2.7
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.087	6.06

7.2.5 AVP In A Manual Test Configuration

7.2.5.1 Description

A verification program developed using an AVP design philosophy targeted for a Manual Test Configuration would force modularization in the following manner. The verification program will consist of an 8K supervisor program and a 24K data buffer. The supervisor program's role is to process test cases resident in the data buffer, calling the vendor's print routine to output results as they are generated. Test cases will be made up of four types of statements (80 character records):

a. ID statement - classifying the type of test.

b. Set up statements - specifying the initialization values.

c. Execute statement - specifying the instruction sequence to be executed.

d. Result statements - specifying the expected values to be compared to the generated results.

Test cases will average about 10 statements in length occupying 400 words of memory. A library system resident on the development system would be used to handle the test case data base. The library system allows edit and file capabilities. The 24K test case data buffer would then be large enough to contain an average of 61 test cases. Therefore, there would have to be 82 distinct program loads under the manual test configuration to process the 5000 expected test cases generated. The test cases would occupy 2 million words of auxiliary storage. It should be noted that the large amount of storage required for this approach is due to the overhead involved in having 80-character control cards of which only ten to twenty percent contains meaningful information. This implies that certain implementation strategies could compress unused blanks, although (for this analysis) a tradeoff like this will not be considered.

The vendor would be responsible for the generation of the load tapes necessary to handle 82 loads. The Air Force would be responsible for sending the 2 megawords of data to the vendor.

7.2.5.2 Non-Recurring Start-Up Costs

The cost for implementing a verification program based on the AVP test approach would be strictly software development costs. The cost breakdown is as follows:

<u>Test Case Development</u>

25 test cases per instruction * 200 instructions	=	5,000 test cases
2 test cases per day productivity	=	2,500 days
250 days/man year	=	10 man years
Supervisor Program Development (3K Machine Language Instructions)	=	1.3 man years
Library System on Development Computer	=	N.C.
Tape Generation Program on Development Computer (500 Machine Language Instructions)	=	0.3 man years
Test Plan Document	=	0.3 man years
TOTAL		11.9 man years

7.2.5.3 Recurring Costs

The *recurring costs* associated with the AVF approach under the Manual Test Configuration are proportional to the staffing allocated during the vendor certification process, and the time it takes to complete the certification. A second cost, though nominal in nature is the cost of generating the test case tape for the vendor prior to starting the in-house test procedure. The final recurring cost component is the cost to sustain the software.

7.2.5.4 Time Required to Perform Validation

The time required to perform the entire certification consists of the summation of mount times, memory load times, and execution times. Assuming 5 minutes to mount the tape, 3 minutes to load memory and 2 minutes to process the 61 tests cases in each program load and print out the results. The maximum time to run the certification tests error free would be calculated as follows:

Verification time = N_1 * 5 minutes to mount each tape +

N_2 * 3 minutes to load and go +

N_2 * 3 minutes to execute the program and print out results

where:

N_1 = number of tape mounts = 82 worst case

N_2 = number of program loads = 82

Verification time = 82 * 5 + 82 * 3 + 82 * 3

= 902 minutes = 15 hours, 2 minutes

7.2.5.5 Impact to SEAFAC Resources

The implementation of the AVP approach under a Manual Test Configuration would require the use of a development computer with library system, and related MIL-STD-1750 support software (cross assembler and simulator). The development computer would also be necessary to generate the source tapes to give to the vendor. There would be no impact to the development computer during the in-house certification process. SEAFAC personnel would be required to develop and maintain the test cases as well as the supervisor program.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-6.

Table 7-6. Cost Summary for the AVP Apprcach Under A Manual Test Configuraticn

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Ccmputer	0	0
Software		
- MIL-STD-1750 Support Software (Crcss Assembler, Linking Loader, Simulator)	0	0
- Test Cases	10.0	700
- Supervisor Program	1.3	91
- Library System on Development Computer	0	0
- Source Tape Generation Program	0.3	21
Other		
- Test Plan Document	0.3	21
TOTAL	11.9	833
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance Supervisor Program: 3K * 2 errors/K * $1,400/30 Test Cases: 2,000K * 0.1 errcrs/K * $1,400/30	0.136	9.5
Personnel		
- Coverage to Observe Executior and Analyze Results	0.4	2.8
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.18	12.58

7.2.6 AVP in a Master/Slave Test Configuration

7.2.6.1 Description

A verification program developed using an AVP design philosophy targeted for a Master/Slave Test Configuration would result in a completely automated verification approach. The verification program would be segmented into an 8K Supervisor Program with a 24K test case data buffer. A bootstrap program would communicate with a control program on the Master computer to first load the Supervisor program into the unit under test (Slave). The supervisor would then automatically roll in test cases and output results as the test cases were executed. Test case size and layout would be the same as in the Manual Test Configuration and are described previously.

The library system on the Master computer would be used to develop the test cases and handle their selection for processing by the control program running during the certification process. The amount of data to be transferred between Master and Slave is as follows:

Test Cases	2,000K words	
Supervisor	8K	
Results	5K	

TOTAL	2,013K words	

The Air Force would send the vendor a copy of the bootstrap program, and the vendor would bring it in IFI format at the time of certification.

7.2.6.2 Non-Recurring Start-Up Costs

The costs for implementing a verification program based on the AVP test approach and an automatic test configuration would be strictly software development costs. The cost breakdown parallels the manual test configuration as follows:

Test Case Development

5,000 test cases at 2 tests cases per day	=	10.0 man years
Control Program Development (Master) (2K HOL Instructions; 500 Machine Language Instructions)	=	1.3 man years
Bootstrap Program Development (Slave) (400 Machine Language Instructions)	=	0.3 man years
Supervisor Program Development (Slave) (3K Machine Language Instructions)	=	1.1 man years
Library System on Master Computer	=	N.C.
Source Tape Generation Program (500 Machine Language Instructions)	=	0.3 man years
I/O Test Programs (1K BAL)	=	0.5 man years
Test Plan Document	=	0.3 man years
	TOTAL	13.8 man years

7.2.6.3 Recurring Costs

The recurring costs associated with the AVP approach under the Master/Slave Test Configuration are associated with system integration and initialization. These costs should not be greater than 1 man day total. The other major costs would be attributed to staffing during the vendor certification process and the cost of sustaining the verification software. A minimal cost due to generating the source tape for delivery to each vendor is also incurred.

7.2.6.4 Time Required to Perform Validation

The time required to perform the complete verification test consists of the summation of Bootstrap Load and GO time, Master Computer Control Program Initiation time, Test Case Execution time and I/O Transfer time. An estimate of these times are as follows:

Test Case Execution Time

```
        5,000 test cases * 10,000
          instructions/test case
          overhead                    = 5,000,000 instructions
     5,000,000 instructions/500,000
          instructions/sec            = 10 seconds
```

I/O Transfer Time

2013K words / 30K words/sec (over MIL-STD-1553) = 67 seconds

```
Verification Time = Bootstrap Load and Go (5 minutes)      +

                    Master Computer Control Program Initialization
                      (5 minutes)                          +

                    Test Case Execution Time (10 seconds)  +

                    Print Results (90 seconds)             +

                    I/O Transfer Time (67 seconds)

                  = 12 minutes, 47 seconds.
```

7.2.6.5 Impact to SEAFAC Resources

The implementation of the AVP approach under a Master/Slave Test Configuration would rely heavily on the Master computer during the development phase of the verification program, and during the actual running of the certification process. The Master computer would need to support a library system as well as the usual selection of support software (assembler, linking loader and simulator). SEAFAC personnel would be required to develop and modify the test cases, bootstrap supervisor and control programs.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-7.

Table 7-7. Cost Summary for the AVP Approach Under A Master/Slave Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.3	91
- Test Cases	10.0	700
- Supervisor Program	1.1	77
- Library System on Master/Development Computer	0	0
- I/O Test Program	0.5	35
Other		
- Test Plan Document	0.3	21
	----	---
TOTAL	13.8	966
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance Control Program: 7.4K * 2 errors/K * $1,400/30 Test Cases: 2,000K * 0.1 errors/K * $1,400/30	0.143	10.02
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
	-----	----
TOTAL	0.191	14.38

7.2.7 Diagnostic Modification in a Manual Test Configuration

7.2.7.1 Description

A verification program based on modifying a diagnostic program or programs for the MIL-STD-1750 to run under a Manual Test Configuration closely resembles the end results from modifying an existing FTP into a certification program as is described subsequently. The development process would require the analysis of an existing diagnostic program or programs, to determine the amount of coverage each instruction is given. The next step would be to remove implementation dependent code, add supplemental test cases, and modify the control structure of the diagnostic program to facilitate dynamic display of results on the printer.

The verification approach would be modularized according to the initial modularization of the diagnostic program selected for modification. A completely new modularization scheme could be imposed oriented to a supervisor module and several test modules, each containing test cases verifying a certain type of instruction or architectural feature of the machine. The estimated size of the verification program is 96K words, thus requiring three distinct program loads. The Air Force would be responsible for making the source available to the vendor for his in-house generation of IPL tapes.

A risk factor, associated with the modification of different vendor's diagnostic programs, concerns the potential incompatibility of assembler language grammar formats, thereby requiring a translation process to occur.

7.2.7.2 Non-Recurring Start-Up Costs

The cost of implementing a certification program based on the modification of existing diagnostic program or programs under a Manual Test Configuration would consist of the following software development components:

Diagnostic Modification (Modify 18K BAL; Write 5.5K BAL)	=	7.0 man years
Tape Generator Program	=	0.3 man years
Test Plan Document	=	0.3 man years
TOTAL		7.6 man years

7.2.7.3 Recurring Costs

The recurring costs associated with the diagnostic modification under a Manual Test Configuration are proportional to the staffing allocated during the vendor verification process, and the actual time it takes to complete the verification. A second cost, though nominal in nature is the cost of generating the verification program source tape for each vendor. The third component is the cost of sustaining the verification software programs.

7.2.7.4 Time Required to Perform Validation

The time required to perform the entire verification can be calculated using the formula developed previously.

Verification time = N_1 * 5 minutes to mount each tape +
N_2 * 3 minutes to load and go +
N_2 * 8 minutes to execute the program and print out the results

where:

N_1 = number of tape mounts = 3 worst case

N_2 = number of program loads = 3

Therefore the Verification time = 3 * 5 + 3 * 3 + 3 * 8
= 48 minutes

7.2.7.5 Impact to SEAFAC Resources

The implementation of a verification program based on the modification of an existing diagnostic program or programs to run under a Manual Test Configuration would require the use of MIL-STD-1750 support software (cross assembler, linking loader, and simulator) on the development computer system. SEAFAC personnel would develop and maintain the certification program as well as prepare the source tape to give to each vendor. During the verification process, there would be no impact on the development computer, but SEAFAC personnel would be required to assist in the integration, initiation, and observation of the verification program executing on the unit under test.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-8.

Table 7-8. Cost Summary for the Diagnostic Approach Under A Manual Test Configuration"

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Computer	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Diagnostic Program Modification	7.0	490
- Source Tape Generation Program	0.3	21
Other		
- Test Plan Document	0.3	21
TOTAL	7.6	532
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 40K x 2 errors/K * $1,400/30	0.053	3.7
Personnel		
- Coverage to Observe Execution and Analyze Results (2 People for 1 Week)	0.04	2.8
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.097	6.78

7.2.8 Diagnostic Modification in a Master/Slave Test Configuration

7.2.8.1 Description

A verification program based on modifying a diagnostic program or programs for the MIL-STD-1750 under a Master/Slave Test Configuration would closely parallel a verification program based on the modification of an existing FTP to run under a Master/Slave Test Configuration as is subsequently described. The development process would require the analysis of the diagnostic programs so that the proper test cases could be added (to increase the coverage of the test) and deleted (to eliminate implementation dependencies inherent to all diagnostic programs). The control structure of the diagnostic program would also have to be modified to facilitate communication with the Master computer.

The approximate size of the verification program would be 96K, divided into a supervisor program and test modules as described previously. The supervisor program would communicate with a control program on the Master computer, requesting test modules to be transferred and sending back test results. The control program on the Master computer would route output messages from the unit under test to a hardcopy device, and load test modules from auxiliary storage to send to the unit under test upon request. A bootstrap load program would also have to be developed with the source being made available to the vendor prior to in-house testing.

7.2.8.2 Non-Recurring Start-Up Costs

The cost of implementing a verification approach based on the modification of existing MIL-STD-1750 diagnostic programs under a Master/Slave Test Configuration would consist of the following software development components:

Diagnostic Modification (Modify 18K BAL; Write 5.5K BAL)	=	7.0 man years
Control Program (Master) (2K HOL; 500 BAL)	=	1.1 man year
Bootstrap Program (400 BAL)	=	0.3 man years
Source Tape Generator Program (500 BAL)	=	0.3 man years
I/O Test Program (1K BAL)	=	0.5 many years
Test Plan Document	=	0.3 man years
TOTAL		9.5 man years

7.2.8.3 Recurring Costs

The recurring costs associated with modifying a diagnostic program to run under a Master/Slave Test Configuration consist of computer usage on the Master computer (zero cost) and staffing for the integration, initiation, observation of the verification program and analysis of the results. A second component is the cost of sustaining the verification program.

7.2.8.4 Time Required to Perform Validation

The time required to perform the complete verification process is calculated by finding the summation of the Bootstrap Load and Go time, Master Computer Control Program Initiation time, Test Case Execution time and I/O transfer time. An estimate of these times follows:

Verification Program Execution
Plus Logging Results = 30 sec

I/O Transfer Time
96K words / 30K word/sec
(over MIL-STD-1553) = 3 sec

Verification time = Bootstrap Load and Go (5 minutes) +

Master Computer Control Program
Initialization (5 minutes) +

Verification Program Execution
Time (30 minutes) +

I/O Transfer Time (3 seconds)

= 40 minutes, 3 seconds.

7.2.8.5 Impact to SEAFAC Resources

The implementation of a verification program by modifying existing diagnostic programs to run under a Master/Slave Test Configuration would utilize the support software (cross assembler, linking loader, and simulator) during program development. During the actual verification process, the Master computer would be necessary to handle the control program function as well as the documentation of the test results. SEAFAC personnel would be required to develop and modify the verification program, bootstrap and control programs.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-9.

Table 7-9. Cost Summary for the Diagnostic Approach Under A Master/Slave Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.1	77
- Diagnostic Modification	7.0	490
- I/O Test Program	0.5	35
Other		
- Test Plan Document	0.3	21
	---	---
TOTAL	9.5	665
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 44.4K * 2 errors/K * $1,400/30	0.059	4.14
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
	-----	----
TOTAL	0.107	7.5

7.2.9 FTP in a Manual Test Configuration

7.2.9.1 Description

A verification program based on the FTP design philosophy targeted for a Manual Test Configuration could be developed in two different ways. The first is to use an existing MIL-STD-1750 FTP to modify its control structure to facilitate dynamic display of results on the printer and to analyze and augment its test cases to achieve 25 test cases per instruction average. The second approach is to implement an FTP type program through the entire software development process.

This type of verification program would be divided into a supervisor module and several test modules. Each test module would contain test cases verifying a certain type of instruction or architectural feature of the machine. The supervisor program is designed and coded under the assumption that no instructions work. This is implemented through utilizing a small core set of instructions to handle control and slowly introducing more instructions as confidence in their credibility is established (center out approach). The supervisor program invokes each test module and outputs test results.

The estimated size of the supervisor and test modules is 96K, thus requiring 3 distinct , program loads. The Air Force would be responsible for making this source available to the vendor for his in-house generation of the IPL tapes.

7.2.9.2 Non-Recurring Start-Up Costs

The cost of implementing a certification program based on the FTP test approach would depend on whether or not the program development was based on the modification of an existing FTP. The software cost breakdown is as follows:

FTP Modification (Modify 30% of 30K lines of BAL = 9K) (Write 1K BAL lines for Control Program)	=	3.1 man years
FTP Full Development (40K BAL instructions)	=	12.0 man years
Tape Generator Program (500 BAL instructions)	=	0.3 man years
Test Plan Document	=	0.3 man years
TOTAL		3.7 man years/ 12.6 man years

7.2.9.3 Recurring Costs

The recurring costs associated with the FTP approach under a Manual Test Configuration are proportional to the staffing allocated during the vendor verification process and the actual time it takes to complete the verification. A second cost, though nominal in nature, is that of generating the FTP source tape for the vendor. The cost of sustaining the verification software programs is the third component.

7.2.9.4 Time Required to Perform Validation

The time required to perform the entire verification consists of the summation of mount times, memory load times and execution times. Assuming 5 minutes to mount the tape, 3 minutes to load memory and 8 minutes to process the test modules on each load tape and print out the results, the maximum time to run the verification test error free would be calculated as follows:

Verification time = N_1 * 5 minutes to mount each tape +

N_2 * 3 minutes to load and go +

N_2 * 8 minutes to execute the program and print out results

where:

N_1 = number of tape mounts = 3 worst case

N_2 = number of program loads = 3

Verification time = 3 * 5 + 3 * 3 + 3 * 8

= 48 minutes

7.2.9.5 Impact to SEAFAC Resources

The implementation of the FTP approach under Manual Test Configuration would require the use of MIL-STD-1750 support software (cross assembler, linking loader and simulator). The development computer would also be necessary to generate the source tape to give to the vendor. There would be no impact to the development computer during the in-house certification process. SEAFAC personnel would be required to develop and maintain the certification program.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-10.

Table 7-10. Cost Summary for the FTP Approach Under A Manual Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Computer	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- FTP Modification	3.1	217
- FTP Full Development	12.0	840
- Source Tape Generation Program	0.3	21
Other		
- Test Plan Document	0.3	21
	-----	----
TOTAL BY MODIFYING FTP	3.7	259
TOTAL BY DEVELOPING NEW FTP	12.6	882
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 40K * 2 errors/K * $1,400/30	0.053	3.7
Personnel		
- Coverage to Observe Execution and Analyze Results (2 People for 1 Week)	0.04	2.8
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
	-----	----
TOTAL	0.097	6.78

7.2.10 FTP in a Master/Slave Test Configuration

7.2.10.1 Description

A certification program developed using a FTP design philosophy targeted for a Master/Slave Test Configuration would result in a completely automated certification approach. The certification program would be approximately 96K in size and be divided into a supervisor module and several test modules. Each test module contains test cases verifying a certain type of instruction or architectural feature. The supervisor program would be designed using the center out approach as previously described. The supervisor program would communicate with a control program resident on the Master computer, requesting test modules to be transferred, and sending back test results. The control program on the Master computer would route output messages from the unit under test to a hardcopy device, and pull in test modules from auxiliary storage to send to the unit under test upon request. The certification program could be developed in two different ways. The first is to take an existing FTP, and modify its control structure to communicate with the control program on the Master computer. The second approach would be to implement an FTP type certification program through the entire software development process. A bootstrap load program would also have to be developed with the source being made available to the vendor prior to in-house testing.

7.2.10.2 Non-Recurring Start-Up Costs

The cost of implementing a certification approach based on the FTP test approach would depend on whether or not the program development was based on the modification of an existing FTP (or whether one is available to modify). The software cost breakdown is as follows:

FTP Modification (Modify 30% of 30K lines of BAL = 9K) (Write 1K BAL lines for Control Program)	=	3.1 man years
FTP Full Development (40K BAL instructions)	=	12.0 man years
Control Program Development (Master) (2K HOL; 500 BAL instructions)	=	1.1 man years
Bootstrap Program Development (Slave) (400 BAL instructions)	=	0.3 man years
Source Tape Generator Program (500 BAL instructions)	=	0.3 man years
I/O Test Programs (1K BAL instructions)	=	0.5 man years
Test Plan Document	=	0.3 man years
TOTAL		5.6 man years/ 14.5 man years

7.2.10.3 Recurring Costs

The recurring costs associated with an FTP approach under the Master/Slave Test Configuration are associated with system integration and initiation. These costs should not be greater than 1 man day total. The other costs would be attributed to staffing during the vendor certification process, and the cost for sustaining the verification software programs.

7.2.10.4 Time Required to Perform Validation

The time required to perform the complete verification tests is calculated by finding the summation of the Bootstrap Load and Go time, Master Computer Program Initiation time, Test Case Execution time and I/O Transfer time. An estimate of these times are as follows:

Test Case Execution Time
5,000 test cases * 50 instructions/test case = 250,000 instructions
250,000 instructions / 500,000 instructions/second = 0.5 seconds

I/O Transfer Time
96K words / 30K words/second (via MIL-STD-1553) = 3.0 seconds

Verification time = Bootstrap Load and Go (5 minutes) +

Master Computer Control Program Initialization (5 minutes) +

Validation Execution Time (0.5 seconds) +

I/O Transfer Time (3.0 seconds)

= 10 minutes, 3.5 seconds.

7.2.10.5 Impact to SEAFAC Resources

The implementation of the FTP approach under a Master/Slave Test Configuration would utilize the support software resident on the Master computer during program development, and during the actual verification process the Master computer would be necessary to handle the control program function as well as documentation of test results. SEAFAC personnel would be required to develop and modify the verification program, bootstrap and control program.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-11.

Table 7-11. Cost Summary for the FTP Approach Under A Master/Slave Test Configuration

	Cost	
Item	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.1	77
- FTP Modification	3.1	217
- FTP Full Development	12.0	840
- I/O Test Programs	0.5	35
Other		
- Test Plan Document	0.3	21
TOTAL BY MODIFYING FTP	5.6	392
TOTAL BY DEVELOPING NEW FTP	14.5	1015
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance 44.4K * 2 errors/K * $1,400/30	0.059	4.14
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
TOTAL	0.107	7.5

7.2.11 Lockstep in a Manual Test Configuration

7.2.11.1 Description

The Lockstep approach is not suitable for implementation under a Manual Test Configuration. A verification program developed using the Lockstep philosophy could be implemented if certain major modifications were made to the initial approach. Two approaches will be described in this section; a semi-Lockstep approach where the trace interrupt is present (henceforth referred to as the SLT approach), and a semi-Lockstep approach where the trace interrupt is not present (henceforth referred to as the SLNT approach). Both approaches execute sequences of instructions referred to as test buckets with trace information being stored sequential in a data buffer area.

The trace interrupt collects the following information:

	WORDS
16 General Purpose Registers	16
Instruction Counter	1
Status Word	1
Fault Register	1
Interrupt Mask	1
TOTAL	20 words

Under the SLT approach, the trace information is automatically collected and saved by the hardware. Under the SLNT approach, the trace information is generated through a software implemented trace routine. When a sequence of instructions has finished execution the generated trace information is compared against predetermined trace information. Each approach contains the following modules:

- Supervisor = 8K words
- Test buckets
  5,000 tests cases * 10 words/test = 50K words
- Predetermined Trace Results
  50,000 cases * 20 words trace information = 1,000K words
- Trace Buffer Area = 1K words

The SLNT has the additional requirement of having a software trace module. The supervisor program executes the test bucket, compares the generated trace information to the expected trace information, and prints out the results.

The size of the trace buffer area directly affects the size of the test buckets, and, the number of test buckets that can be contained in one program load. The following equation expresses this relationship:

Memory size = Supervisor size + Trace Buffer size +

Test Bucket area + Predetermined Trace Information

where:

Test Bucket area = Number of Test Buckets * Test Bucket Size
Predetermined Trace Information = Number of Test Buckets * Test Bucket Size * 20

$$\text{Test Bucket Size} = \frac{\text{Trace Buffer Size}}{\text{20 words per instruction}}$$

For a 32K machine, we are proposing an 8K supervisor and a 1K trace buffer.

Let NTB = Number of Test Buckets
TBS = Test Bucket Size

Therefore:

$$TBS = \frac{\text{1024 words}}{\text{20 words per instruction}}$$

$$= \text{51 instructions per bucket}$$

And substituting back into the top equation

$$32K = 8K + 1K + NTB * 51 + NTB * 51 * 20$$

$$23K = NTD * 51\ (1 + 20)$$

Solving for NTB:

$$NTB = \frac{23K}{50(21)} = \text{22 test cases}$$

It should be mentioned that certain architectural features require a set sequence of instructions to be executed in order to verify the proper execution. Therefore, the size of the test buckets must allow for this factor. Analysis from data gathered on various test approaches suggest that minimal test bucket size should be no less than 30 words.

The number of separate program loads necessary to implement these approaches is calculated to be (50K words in test sequences/(21 test buckets * 51 words/test bucket)) = 46. The Air Force would be responsible for sending 1.1 megawords of data to the vendor for the generation of the 46 IPL tapes. Note, the SLT and SLNT approaches closely resemble the AVP approach.

7.2.11.2 Non-Recurring Start-Up Costs.

The cost of implementing a certification program based on the SLT or SLNT approach would consist of the following software components:

Test Case Development			
5,000 test cases * 10 words/ test case	=	50,000	words
50,000 words/(51 words/test bucket)	=	980	test buckets
1 test bucket + results/ day productivity	=	980	days
250 days/year	=	4	man years
Supervisor Program Development (4K BAL instructions)	=	1.8	man year
Software Trace Module Development (850 BAL instructions)	=	0.4	man years
Tape Generator Program for Source (500 BAL instructions)	=	0.3	man years
Test Program Development	=	0.3	man years
SLT TOTAL		6.4	man years
SLNT TOTAL		6.8	man years

7.2.11.3 Recurring Costs

The recurring costs associated with the SIT and SLNT approaches are proportional to the staffing allocated during verification process, and the time it takes to complete the verification. A second cost is attributed to generating the source load tape for each vendor's IPL tape generation. The third cost component is the cost of sustaining the verification software.

7.2.11.4 Time Required to Perform Validation

The time required to perform the entire verification can be calculated using the following formula:

Verification time = N_1 * time to mount IPL tape +

N_2 * time to load memory + start program +

N_2 * time to execute program load

where:

N_1 = number of tape mounts

N_2 = number of program loads.

Assuming 5 minutes to mount a tape, 3 minutes to load the memory and start the program and 2 minutes to process the 22 test buckets in each program load and print out the results. The maximum time to run the verification occurs when $N_1 = N_2$ (separate mount required for each program load). The verification time to complete an error free execution of the complete program would be:

Verification time = N_1 * 5 minutes to mount each tape +

N_2 * 3 minutes to load and go +

N_2 * 2 minutes to execute the program and print out results

where:

N_1 = number of tape mounts = 46 worst case

N_2 = number of program loads = 46

Verification time = 46 * 5 + 46 * 3 + 46 * 2

= 460 minutes = 7 hours, 40 minutes

7.2.11.5 Impact to SEAFAC Resources

The implementation of the SLT or SLNT approach would require the use of a development computer preferably with a library system to facilitate the test bucket data base. Support software consisting of cross assembler, linking loader and simulator must also be available. The development computer would be used to generate the source tapes to give to the vendor. There would be no impact to the development computer during the in-house verification process. SEAFAC personnel would be required to develop and maintain the test buckets as well as the supervisor program.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-12.

Table 7-12. Cost Summary for the Lockstep Approach Under A Manual Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development Computer	0	0
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Test Cases with Results	4.0	280
- Supervisor Program	1.8	126
- Software Trace Program	0.4	28
- Source Tape Generation Program	0.3	21
Other		
- Test Plan Document	0.3	21
TOTAL WITH HARDWARE TRACE	6.4	448
TOTAL WITH SOFTWARE TRACE	6.8	476
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
Software		
- Maintenance Supervisor: 4.5K * 2 errors/K * $1,400/30 Test Cases: 1,050K * 0.2 errors/K * $1,400/30	0.146	10.22
Personnel		
- Coverage to Observe Execution and Analyze Results (2 People for 1 Week)	0.04	2.8
Other		
- Test Plan to Vendor with Certification Source	0.004	0.28
TOTAL	0.19	13.3

7.2.12 Lockstep in a Master/Slave Configuration

7.2.12.1 Description

A verification program developed using a Lockstep design philosophy is ideally suited for a Master/Slave Test Configuration. This approach can be separated into four different implementations. First, the master computer can be a "golden" MIL-STD-1750 computer. Second, a MIL-STD-1750 simulator running on the development machine can be a "golden" computer. Third, the MIL-STD-1750 under test may or may not have the trace interrupt feature. The compensation for lack of hardware trace has been discussed previously, and it is assumed that a similar software module can be developed in this situation. A fourth possible variation on this approach can occur if we assume that no simulator or "golden" computer exists. Instead, predetermined results are calculated during test case development and these results are compared to the generated trace results. This variation closely resembles the semi-Lockstep approaches (SLT and SLNT) described previously. All approaches execute sequences of instructions referred to as test buckets with trace information stored sequentially in a data buffer area. The trace data collected will consist of 20 words reflecting the state of the computer after the execution of the current instruction (see Section 7.2.11.1 for details). When a sequence of instructions has finished execution, the trace information generated by the unit under test (slave) is compared with the trace information generated by the "golden" computer or simulator, except in the semi-Lockstep approaches where the generated trace data is compared to the expected trace data. The results are then printed out.

The verification program based on a Lockstep approach under a Master/Slave Test Configuration would be divided into the following modules:

- Bootstrap Load Program	=	1K words
- Supervisor	=	8K words
- Test Bucket Data Base	=	50K words
- Predetermined Results (optional)	=	1,000K words
- Simulator (optional)	=	300K words
- Control Program	=	8K words
- I/O Test Programs	=	4K words

The following sequence of events describes the interaction and function of each module. (Note: the semi-Lockstep approach is described later.)

1. The bootstrap load program establishes communication with the control program on the Master computer and starts the transfer of a supervisor program. This process takes place for the unit under test as well as the "golden" computer, if applicable.

2. The supervisor program requests test buckets for execution from the Master computer.

3. The unit under test executes the test buckets, while the "golden" computer or the simulator executes the same instructions.

4. The results from the unit under test and the "golden" computer or the simulator are sent back to the Master computer for comparison.

5. The results are recorded.

6. Steps 1 through 5 are repeated until all test buckets and results have been transmitted, executed and compared.

For the semi-Lockstep approach there is no "golden" computer, or simulator and the sequence of events would flow in similar order with the comparison of generated results to the predetermined expected results being made by the control program on the Master computer.

The following is a breakdown of data transfers taking place via the control program:

Supervisor to Unit Under Test	=	8K	words
Test Cases to Unit Under Test	=	50K	words
Results from Unit Under Test	=	1,000K	words
Supervisor to "Golden" Computer	=	8K	words
Test Cases to [illegible]den" Computer	=	50K	words
Results from "Golden" Computer	=	1,000K	words
TOTAL		2,116K	words

7.2.12.2 Non-Recurring Start-Up Costs

The cost components for implementing a verification approach based on a Lockstep test approach under a Master/Slave Test Configuration would depend upon the final variation of the test approach selected. The software development cost would be as follows:

Control Program (Master) (3K HOL; 1K BAL)	=	1.8 man years
Bootstrap Program (Slave) (400 BAL)	=	0.3 man years
Supervisor Program (3K BAL)	=	1.4 man year
Test Buckets	=	3.0 man years
Test Bucket Results (optional)	=	1.0 man year
Software Trace Module (optional) (850 BAL)	=	0.4 man years
Simulator (optional) (12K HOL; 2K BAL)	=	6.0 man years
Source Tape Generator Program (500 BAL)	=	0.3 man years
I/O Test Programs (1K BAL)	=	0.5 man years
Test Plan Document	=	0.3 man years

The Non-Recurring Start-Up costs for the "golden" computer can be broken down into the hardware procurement cost and the architecture verification process performed to validate the integrity of the "golden" computer. This verification is extremely crucial in assuring the quality of the system and would require detailed analysis almost comparable to any of the other approaches.

The estimates for these costs are as follows:

MIL-STD-1750 Computer Acquisition and Ground Support Equipment	=	$500K
"Golden" Computer Verification	=	2.5 man years

The software totals then for each approach are as follows:

Lockstep with "Golden" Computer	=	10.1 man years
Lockstep with Simulator	=	13.6 man years
Semi-Lockstep, predetermined Results	=	8.6 man years

Note: An additional 0.4 man years would be added to each total if a software trace feature was required to be developed and integrated.

7.2.12.3 Recurring Costs

The recurring costs associated with the lockstep approach and its variations consists of the hardware maintenance cost associated with the "golden" computer, the personnel allocated during the verification process, and the cost of sustaining the verification software programs.

7.2.12.4 Time Required to Perform Validation

The time required to transfer, execute and compare the 5,000 test buckets can be calculated by substituting appropriate constants into the following equation:

Verification time = Bootstrap Load Time for Unit Under Test +
Bootstrap Load Time for "Golden" Computer +
Master Computer Initialization Time +
Test Case Transfer Time to Unit Under Test +
Test Case Transfer Time to "Golden" Computer +
Test Case Execution Time +
Supervisor Program Overhead +
Test Results Transfer from Unit Under Test to Master +
Test Results Transfer from "Golden" Computer to Master +
Time to Compare Trace Information and Print Results.

Substituting line-by-line we have:

```
    Verification time  =  5 min +
                          5 min +
                          5 min +
                          2 sec +
                          2 sec +
                         (50,000 instructions / 500,000 instructions/
                          sec = 0.1 sec) +
                         (50K instructions * 1K overhead factor  /
                          500K instructions/sec = 100 sec) +
                       33.3 sec +
                       33.3 sec +
                         10 min

       Total Time      = 27 minutes, 50 seconds.
```

7.2.12.5 Impact to SEAFAC Resources

The implementation of the Lockstep approach for a verification program under a Master/Slave configuration, would require the use of a library system on the Master computer to control the development of test buckets and potentially their predetermined results. Support software consisting of a cross assembler, linking loader and simulator would also have to be available on the Master. SEAFAC personnel would be required to develop and maintain the test buckets and all software modules that comprise the verification program as well as supervise the certification process each time a vendor brings a box to be tested. If a "golden" computer is selected, then SEAFAC personnel must also develop, verify and maintain it, and its related Ground Support Equipment. Also additional space and power must be made available to accommodate the "golden" computer.

The cost data impact to SEAFAC (and the previously discussed cost data) is summarized in Table 7-13.

Table 7-13. Cost Summary for the Lockstep Approach Under a Master/Slave Test Configuration

Item	Cost	
	Man Years	K $
Non-Recurring Start-Up Costs		
Hardware		
- Development/Master Computer	0	0
- MIL-STD-1553 and RS-232 I/O Interfaces	0	0
- "Golden" Computer	7.14	500
Software		
- MIL-STD-1750 Support Software (Cross Assembler, Linking Loader, Simulator)	0	0
- Bootstrap Load Program	0.3	21
- Source Tape Generation Program	0.3	21
- Control Program on Master	1.8	126
- Supervisor Program	1.4	98
- Test Buckets	3.0	210
- Test Bucket Results	1.0	70
- Software Trace Module	0.4	28
- Simulator	6.0	420
- I/O Test Programs	0.5	35
Other		
- Test Plan Document	0.3	21
- "Golden" Computer Verification	2.5	175
	----	----
TOTAL WITH "GOLDEN" COMPUTER	17.2	1207
TOTAL WITH SIMULATOR	13.6	943
TOTAL WITH PREDETERMINED RESULTS	8.6	602
(NOTE: ADD $28K TO EACH FIGURE IF NO HARDWARE TRACE IS AVAILABLE)		

Table 7-13. Cost Summary for the Lockstep Approach Under a Master/Slave Test Configuration (cont)

	Cost	
Item	Man Years	K $
Recurring Costs/Computer		
Hardware		
- Maintenance	0	0
- "Golden" Computer Maintenance	0.086	6.0
Software		
- Maintenance		
"Golden" Computer: 58K * 2 errors/K * $1,400/30	0.077	5.41
Simulator: 71K * 2 errors/K * $1,400/30	0.096	6.72
Predetermined Results:		
1,051K * 0.2 errors/K * $1,400/30	0.14	9.81
Personnel		
- Coverage to Initialize, Observe and Analyze Results (2 People for 1 Week)	0.04	2.8
- Technician to Supervise Integration of I/O Interface	0.004	0.28
Other		
- Test Plan to Vendor with Verification Source	0.004	0.28
"GOLDEN" COMPUTER TOTAL	0.211	14.77
SIMULATOR TOTAL	0.144	10.08
PREDETERMINED RESULTS TOTAL	0.188	13.17

7.3 QUALITY OF VERIFICATION APPROACHES AND SOFTWARE VALIDATION COSTS

A discussion of the concepts behind the quality of verification approaches and the methodology used to measure that quality and its associated costs was described in earlier sections.

This section describes the application of that methodology. It is presented in three parts. The first part describes the raw data gathered and the techniques used to gather that data. The second part analyzes and summarizes this data. Finally, the third part presents an interpretation of the results.

7.3.1 Data Collection

Two distinct steps have been identified as necessary for estimating the costs to be assigned to the different verification approaches. These have been discussed in detail previously. They are summarized here:

Step 1 Establish the relationship between the number of architectural discrepancies remaining in a computer after architectural verification (sell-off) and the cost of permitting those discrepancies (because of an imperfect verification method).

Step 2 For each verification method, determine the number of architectural discrepancies expected to remain after verification, and, using the relationship established in Step 1, compute the cost associated with those remaining architectural discrepancies.

7.3.1.1 Data Collection for Step 1

Five avionics programs were identified in the proposal to the Air Force as potentially useful for providing data for Step 1. These were:

PAVE LOW
A-7
B-52D SPN/GEANS
F-111
E-3A AWACS

In the course of study, nine other potentially useful programs have been identified. Each of these is a hardware upgrade. (See Section 5.2.1.3 for a discussion of the significance of the hardware upgrade aspect.) These programs are:

PAVE TACK
PAVE TACK/VATS
F-111
Space Shuttle
A-6E
A-6 Universal Missile Wiring System
LAMPS
EA-6B
F-8

All fourteen programs were investigated to obtain a general understanding of each. Data collected included the mission, customer, computers involved, roll-off date, nature of change, and the names of the hardware, software, configuration control, and Program Office personnel.

These programs were then scrutinized for applicability of data. Eight of the fourteen programs were eliminated. Various reasons apply. For example, the Space Shuttle software validation costs were not deemed representative of the validation costs which would be expected for the anticipated MIL-STD-1750 applications because of the special reliability concerns due to large investments in the program and redundancy testing necessary for manned safety. As another example, the A-6 Universal Missile Wiring System was only in the final checkout stage at the time of data collection. Hence, the data is not available.

The avionics upgrade programs which remain are:

B-52D SPN/GEANS
A-7
PAVE TACK
PAVE TACK/VATS
F-111
PAVE LOW

The computers involved in these programs are shown in Table 7-14. Detailed software validation data and EC data were then collected for these programs. These data are listed in Tables 7-15 through 7-20, respectively.

Table 7-14. Hardware Upgrades

Program	Sell-Off Date	Original Computer/ Part Number	New Computer/ Part Number
A-7	10/15/76	TC-2A(74) 6870600 (Pre-Prod)	TC-2A (76) 6870500-1 (Prod)
PAVE LOW	01/26/79	TC-2A (76) 6870500-2	TC-3B 6259000-1
PAVE TACK	04/01/76	TC-3 6870400-1	TC-3 6870400-1
PAVE TACK/VATS	02/28/78	TC-3 6870400-1 (Pre-Prod)	TC-3A 6870400-2 (VATS)
F-111	02/17/78	CP-2	CP-2A 6195500-1
B-52D SEN/GEANS	2/24/79	AP-101 6160025	AP-101C 6217800-20

Table 7-15. A-7 Program Data

Program:	A-7
Customer:	Navy/LTV
Computers:	TC-2A (74), converted to TC-2A (76)
Sell-Off Date:	15 October 1976
Operational Flight Program Validation:	Naval Weapons Center, China Lake

Function Description: Navigation and weapons delivery for Close Support Light Attack aircraft.

Hardware Change Summary: Changed from Modular Core Memory (MCM) to Double Density Modular Core Memory (DMCM), increasing maximum memory capacity from 40K by 16 bits to 64K by 17 bits. Added memory parity, hardened base registers, and five instructions. Optimized I/O logic and increased CPU performance. Added surge amp in converter for wheels drive.

Software Revalidation Data: The navigation and weapons delivery program was revalidated by the Navy at the China Lake Naval Weapons Center. The program was 16K in length. Effort expended was 96 man-weeks. The initial validation effort was performed on a laboratory simulator, then 30 flight tests were performed.

EC Data: Twenty-four architecture related changes have been made since sell-off. (See the discussion about the scope of "architectural relevance" in the Section 5.2.1.2.) These ECs are plotted in Figure 7-4.

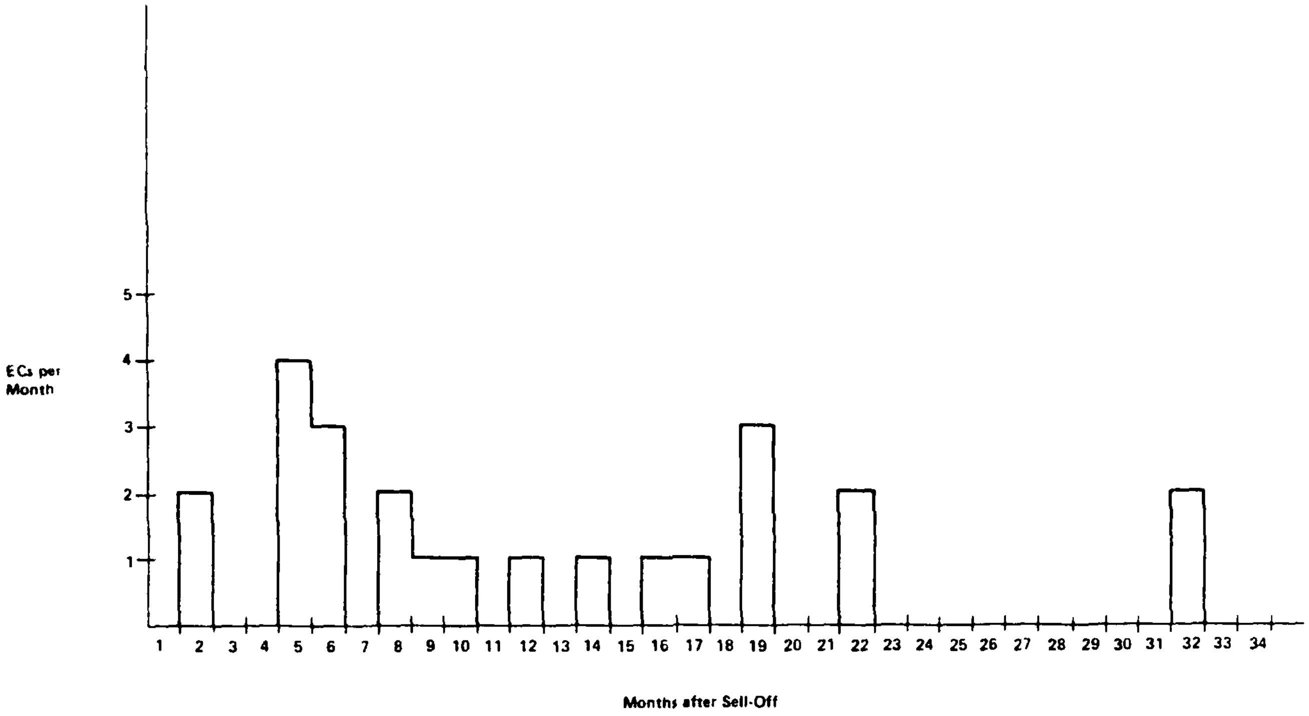


Figure 7-4. ECs Since Sell-Off for the A-7 Program
(Sheet 1 of 1)

Table 7-16. B-52D SPN/GEANS Program Data

Program: B-52D SPN/GEANS

Customer: Air Force/Warner Robbins Air Logistics Center

Computers: AP-101, converted to AP-101C

Sell-Off Date: 12 March 1979

Operational Flight Program Validation: IBM

Function Description: The computer is used to control an Inertial Measuring Unit (platform) for precision navigation of the B-52D. This unit is designated SPN/GEANS (Standard Precision Navigation/ Gimballed Electrostatically Suspended Gyro Airborne Navigation System).

Hardware Change Summary: The AP-101 computer was replaced with the AP-101C. This involved a complete re-packaging of the machine, including CPU, I/O, and memory.

Software Revalidation Data: Software revalidation began on 12 March 1979. Approximately 1/3 of the effort was for adding new functions; the data has this taken into account. Flight testing has not begun. Therefore, estimates of the total expected number of flight tests have been made by personnel experienced in flight testing. Projected effort is 43 man weeks. Approximately twelve flight tests are expected. The size of the program being revalidated is 18K.

EC Data: Twenty-six ECs of architectural relevance have been written since sell-off. (See the discussion about the scope of "architectural relevance" in Section 5.2.1.2.) These ECs are plotted in Figure 7-5.

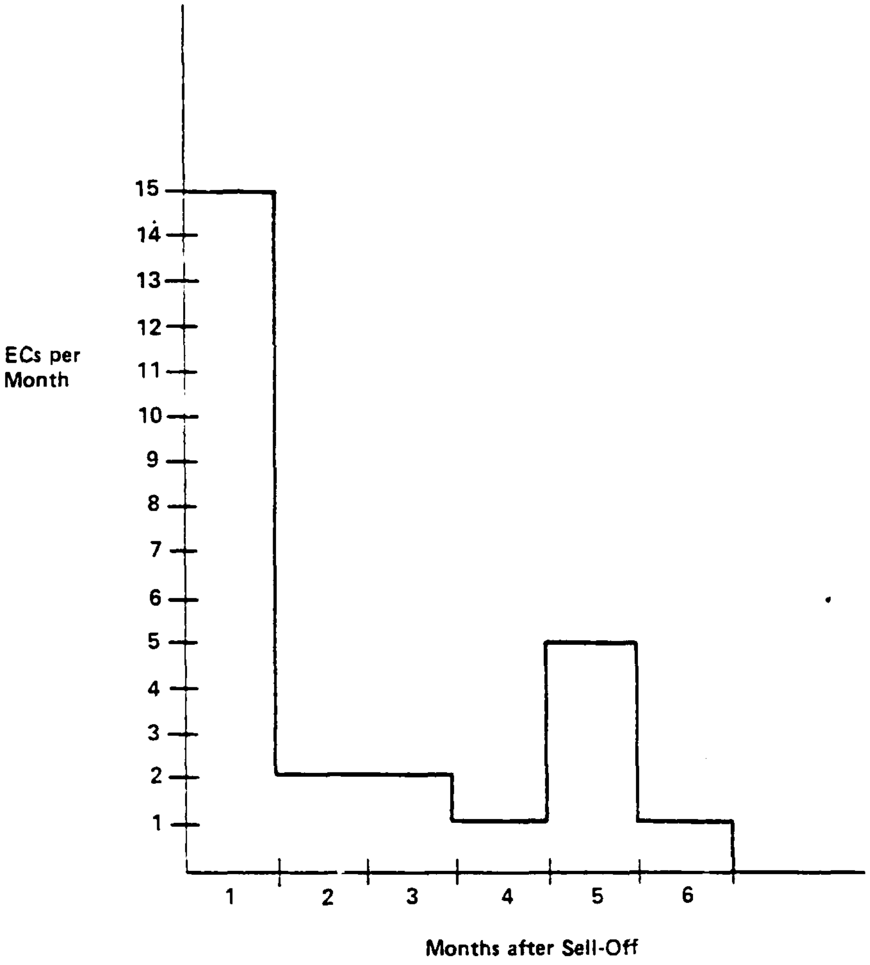


Figure 7-5. ECs since Sell-Off for the B-52D SPN/GEANS Program
(Sheet 1 of 1)

Table 7-17. PAVE TACK Program Data

Program: PAVE TACK

Customer: Air Force/Ford Aerospace

Computers: TC-3

Sell-Off Date: 1 April 1976

Operational Flight Program Validation: IBM

Function Description: The function is to identify targets for precision electro-optical weapons using a laser ranger/designator pod. A TV displays target information to operator and a pod interfaces with the NAV computer. This is used on both the F-4 and F-111 platforms.

Hardware Change Summary: The memory was doubled in size from 8K to 16K, by converting from one MCM to two MCM.

Software Revalidation Data: Two forms of self-test facilitated checkout. The first was a minimal test which is invoked periodically during the execution of the Operational Flight Program. The second was a Self Test Mode which drives the pod hardware and feeds back data for a closed-loop check.

The program which was revalidated was 8K in size and took one man week plus one flight test to revalidate.

EC Data: One EC of architectural relevance was written after sell-off (1 April 1976).

Table 7-18. PAVE LOW Program data

Program: PAVE LOW

Customer: Air Force/ASD/SDX

Computers: TC-2, converted to TC-3B

Sell-Off Date: 26 January 1979

Operational Flight Program Validation: IBM

Function Description: The PAVE LOW mission is a helicopter search and rescue during night/adverse weather. The equipment performs precision navigation and locates survivors. The computer estimates survivor latitude and longitude (based on "beeper" information) and manages a platform, IR sensors, and terrain following radar.

Hardware Change Summary: The unique PAVE LOW I/O was incorporated into the computer LRU. Processor speed was increased. Memory size increased from 16K to 32K by replacing the Basic Operating Memory with DMCM. New instructions were added and the logic was optimized.

Software Revalidation Data: Revalidation effort consisted of four man days of lab effort and one flight test. The program which was revalidated was 16K in length.

EC Data: Only one architecture related EC was written since sell-off on 26 January 1979.

Table 7-19. PAVE TACK/VATS Program Data

Program: PAVE TACK/VATS

Customer: Air Force/Ford Aerospace

Computers: TC-3

Sell-Off Date: 28 February 1978

Operational Flight Program Validation: IBM

Function Description: The function is to identify targets for precision electro-optical weapons using laser ranger/designator pod. A TV displays target information to operator and a pod interfaces with the NAV computer. This is used on both the F-4 and F-111 platforms.

Hardware Change Summary: Additional I/O channel capability was added to interface with the Video Augmented Tracker System. This allowed a 50 KHz input channel to be multiplexed between the laser ranger and the Video Tracking Unit.

Software Revalidation Data: Two forms of self-test facilitated checkout. The first was a minimal test which is invoked periodically during the execution of the Operational Flight Program. The second was a Self Test Mode which drives the pod hardware and feeds back data for a closed-loop check.

The program which was revalidated was 12K in size and took two man-weeks plus one flight test to revalidate.

EC Data: No engineering changes reflecting architecture deviations were written since initial sell-off (2/28/78).

Table 7-20. F-111 Program Data

Program: F-111

Customer: Air Force/SMALC

Computers: CP-2, converted to CP-2A

Architecture Verification Date: 17 February 1978

Operational Flight Program Validation: Sacramento Air Logistics Center, Engineering Division

Function Description: The F-111 is a tactical fighter-bomber for deep interdiction and strategic missions. The computers perform navigation and weapons delivery.

Hardware Change Summary: The CP-2 machine was completely redesigned and repackaged, while retaining instruction set compatibility. Storage capacity was increased from 16K by 18 bits to 64K by 18 bits. Performance was increased by a factor of three.

Software Revalidation Data: The CP-2A brassboard machine was submitted to the Air Force as part of an unsolicited proposal. This machine was evaluated by validation of proper execution of various F-111 Operational Flight Programs. Tests were conducted at the F-111 Avionics Integration Support Facility, using both static and simulated flight conditions. No flight tests were performed; however, it was estimated that six flight tests would be necessary. Validation was completed for three OFPs which were 20K, 16K, and 16K in length. The 20K program was a composite of the other two. Effort expended for these programs was 54 man weeks.

EC Data: Since the machine which was used for validation was a breadboard, changes were recorded but not formally written up as ECs. Hence, no time information is available. A total of nine architectural discrepancies were found. Testing took place over a 2 1/2 month period. Use of the machine was then discontinued. (Note that this machine was in the form of an unsolicited proposal; it was not sold.)

EC data collection typically began by obtaining an "As-Built List" for the computer in question. This list contains the part numbers for all of the major subassemblies in each computer. Summary EC information was obtained for each major subassembly in the computer via the Owego Part Number Identification User System (CPIUS). This is an automated system which allows a user, via a keyboard/display terminal, to obtain a listing of all ECs written against each part number.

Each EC was physically pulled from the records center and examined to determine its architectural relevance. (See the discussion about architectural relevance in Section 5.2.1.2.) Each EC contains marked-up drawings, wiring lists, parts lists, and a description of the nature of the change. For each EC judged to be relevant, the EC number and date were recorded.

The software revalidation data listed in Tables 7-15 through 7-20 is summarized in Table 7-21. The cost figures, both in terms of total cost and cost per K lines of code, are also shown in Table 7-21. Two cost elements comprise the total. The first is the weekly wage cost, $1,346, which is multiplied by the number of man weeks of effort. The second is the average cost per flight test. Two data points are averaged here: the current average cost used by the Navy at the China Lake Naval Weapons Center which is $2,000 and the anticipated cost of the flight tests for F-111, which was $3,000. The resulting $2,500 average is multiplied by the number of flight tests.

Table 7-21. Operational Flight Program Revalidation Data

		Revalidation Data				
Program	Computer	Program Size	MW	Flight Tests	Total Cost	Cost per K Lines of Code
A-7	TC-2A	16K	96	30	204,216	12,764
PAVE LOW	TC-3B	16K	1	1	3,846	240
PAVE TACK	TC-3	8K	1	1	3,846	481
PAVE TACK/VATS	TC-3A	12K	2	1	5,192	433
F-111	CP-2A	20K, 16K, 16K	54	6*	87,684	1,686
B-52D/SPN GEANS	AP-101C	18K	43	12*	87,878	4,882

*Includes projections.

7.3.1.2 Data Collection for Step 2

The application of Step 2 described above required the collection of EC data for three types of verification approaches:

Functional

Random

Lockstep

For the Functional approach, three of the programs which have already been investigated are applicable. These are B-52D SPN/GEANS, A-7, and F-111. The machines in these programs underwent complete, or nearly complete, changes. As a result, the data should represent the results that could be expected for application of a Functional type verification process to a typical MIL-STD-1750 machine. (Note the deficiencies, however, in the F-111 data, described in Table 7-20.) The EC data in Tables 7-15, 7-16 and 7-20 apply.

For the Random approach, an IBM S/370 Model was investigated. Unfortunately, no machines have been built which were tested solely with the Random approach. In all cases, the Functional method was also applied. The test package consisted of a control program (operating system) and the following tests:

- Storage
- Relocation Architecture
- Operating System Hardware
- Storage Protection
- Miscellaneous I/O

Key to this package is the use of the random instruction generator.

Data gathered were Requests for Engineering Action (REAs) which are equivalent to Engineering Changes. Each REA provides a description of the change and the reason for the change. Each REA was examined as to its relevancy; non-architecture related changes were thrown out.

First ship was March 17, 1978. REAs prior to this date were ignored. Change data are shown in Figure 7-6.

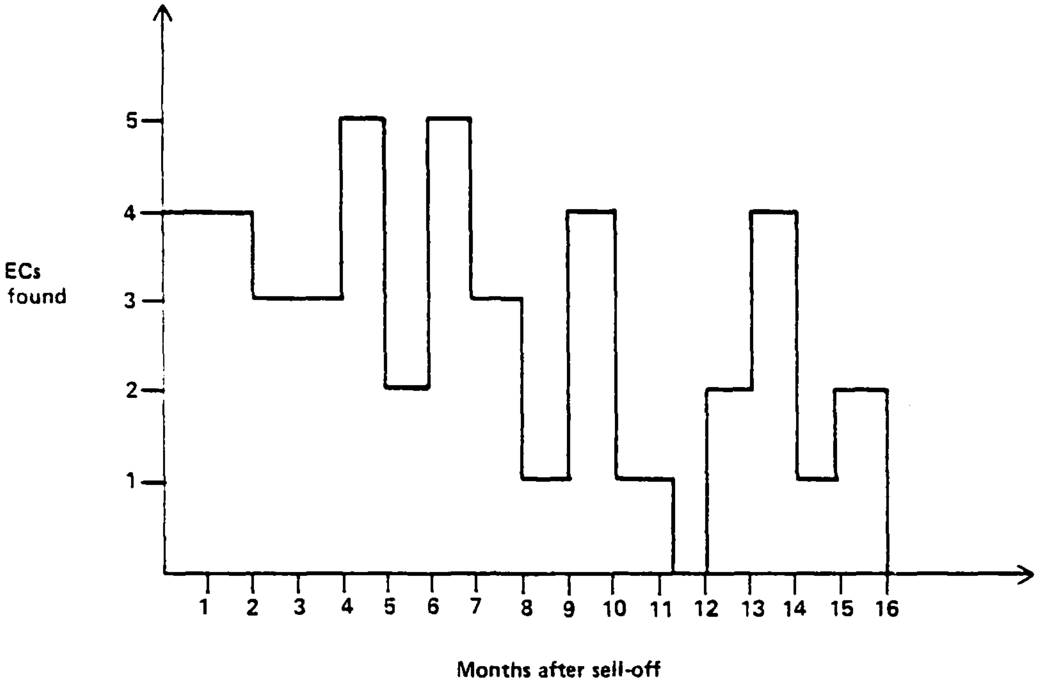


Figure 7-6. ECs of Architectural Relevance for System/370 Model versus Time

For the Lockstep approach, the IBM Series/1 Model 4952 was chosen. This method has been employed by the IBM General Systems Division to test several models of the Series/1 computer: the 4953, 4955, and the 4952. Lockstep had been applied to the first two models subsequent to first ship. Only the Model 4952 had been verified by the Lockstep method prior to first customer ship. First ship date was June 4, 1979. At the time of data collection, August 29, 1979, no Engineering Changes had been written against this model.

Also required for the application of Step 2 was the collection of machine size and EC distribution information for the normalization of the EC data. The methods for application of these data are described in Appendix B.

The machine size data was gathered with the use of the OPIUS system and computer-generated indented parts lists. For upgraded computers, each subassembly parts list was compared for differences between the original and the upgraded machines. These differences were tabulated and totaled to arrive at the machine change size in terms of gates, microcode bits, and main storage bits. These data are shown in Table 7-22.

Table 7-22. Hardware Change Size/Machine Size

Program	Logic (Gates)	Microcode (Bits)	Main Store (Bits)
A-7	15,350	0	458,752
PAVE LOW	7,519	0	0
PAVE TACK	0	0	131,072
PAVE TACK/VATS	4	0	0
F-111	15,715	547,136	1,179,648
B-52D/SPN GEANS	29,894	611,201	1,179,648
Series/1*	-	-	-
System/370 Model	274,000	1,719,296	8,388,608

* Since no ECs were found, no machine size data collection was necessary.

Each relevant EC was examined to determine whether the cause was logic-related, microcode-related or main store-related. These data are shown in Table 7-23.

Table 7-23. Distribution of ECs

Program	Logic	Microcode	Main Store
A-7	16	0	8
PAVE LOW	0	0	1
PAVE TACK	1	0	0
PAVE TACK/VATS	0	0	0
F-111	6	1	2
B-52D/SPN GEANS	16	10	0
Series/1*	-	-	-
System/370 Model	26	14	4

*Since no ECs were found, no machine size data collection was necessary.

The procedure for normalization of architectural discrepancies requires information on the size of the nominal MIL-STD-1750 machine. The nominal size of the MIL-STD-1750 machine is assumed to be the same as the machine used in B-52D SPN/GEANS. This machine, which was recently developed, contained 64K by 32 bits of main storage, two MIL-STD-1553 channels, and discrete I/O. Performance is approximately 500 KOPS. The unit is packaged in a full ATR configuration. This is believed to be typical of the forthcoming MIL-STD-1750 machine.

7.3.2 Analysis of Quality Data

This section contains an analysis of the data described in the previous section. It begins with the estimation of the total number of architectural discrepancies for the programs previously described. This is done with the aid of a set of APL programs which were written to implement the correlation coefficient and squared error techniques described in Appendices B and C.

Next, the relationship between the projected total number of architectural discrepancies remaining after verification and the cost of having those discrepancies remain is established.

Finally, the quality associated with each verification method is discussed.

7.3.2.1 Application of Correlation Coefficient and Squared Error Techniques to EC Data

Engineering Change data for the various programs and machines has previously been described. To summarize the results, sufficient data exist to estimate the total number of architectural discrepancies for three machines/programs: A-7, B-52D SEN/GEANS, and the System/370 Model. Series/1, PAVE LOW, PAVE TACK/VATS, and PAVE TACK have either zero or one EC; no projection is therefore possible. The F-111 data has no time information and the method cannot be applied.

To aid in determining the projected total number of architectural discrepancies for A-7, B-52D SEN/GEANS, and the System/370 Model, APL programs have been written which apply the correlation coefficient and squared error techniques which are described in Appendix B. Five programs have been written:

X FIT Y
F
MEAN
VAR
ESTIMATE

These programs are listed in Appendix C. The program X FIT Y transforms the cumulative data, g(t), and computes the best fitting line using the linear regression formulas given in Appendix B. It calls upon the programs MEAN and VAR to compute the means and variances required for the regression analysis. X FIT Y also computes the correlation coefficient and the squared error. For the latter, it calls upon the program F to calculate the values of the fitted curve, g(t).

Finally, the program ESTIMATE is used to estimate the total number of architectural discrepancies. It does so by successive approximation. It chooses values of C (as previously discussed) and invokes X FIT Y to provide the corresponding correlation coefficients and squared errors. It determines how to change C to obtain either a higher correlation coefficient or a lower squared error. It continues to test different values of C until the best estimate of C is obtained.

Notice that the program ESTIMATE as listed in Appendix C prints "A-7 Engineering Changes". This is changed for each new set of data.

Also note that, as listed, it optimizes based on the squared error. A simple change makes it optimize on the correlation coefficient.

The squared error is a better measure of fit, since it compares the calculated curve, g(x), with the actual data. The correlation coefficient, however, measures the degree of fit of the calculated line with the <u>transformed</u> data. Since the transform is non-linear, error may be introduced. Because of this, the squared error technique is used for estimation of C. However, the calculations are also performed using the correlation coefficient technique, since it is expected that the errors introduced would be small. The results serve as a check of the squared error method, and are provided for information purposes only.

The results of these programs are provided in Appendix D. They are summarized in Table 7-24.

Table 7-24. Estimate of Total Architectural Discrepancies

Program	Total ECs	Estimated Total Architectural Discrepancies	
		Squared Error Method	Correlation Coefficient Method
B-52D SEN/ GEANS	26	27	30
A-7	24	29	28
System/370 Model	44	68	66

It is satisfying to note that the projections using the two different methods track quite . closely. It is also satisfying to note that the projections for the System/370 Model using the two methods compare favorably with the projection of 64 total architectural discrepancies that was obtained previously using the decreasing exponential method.

7.3.2.2 Projections for Programs with Insufficient Data

As was mentioned previously, five of the programs do not have sufficient data to permit use of the foregoing methods. Four of these five have either zero or one EC written against them. Since insufficient data points exist to fit lines for these four programs, the data will be used as is; i.e., the projected total number of architectural discrepancies is the same as the number of ECs found to date.

The fifth program is F-111, which has nine architectural discrepancies but no time information. Because of this, the methods of projection of the total number of architectural discrepancies cannot be applied. Again the data is used as is; nine architectural discrepancies is the projected total for F-111. This necessarily biases the data, but there is another, offsetting, bias. This will be discussed more fully in Section 7.3.2.4.

7.3.2.3 Estimation of the Cost vs Architectural Discrepancies Relationship

The cost of validation of Operational Flight Programs (OFPs) versus the projected number of architectural discrepancies remaining after verification is plotted in Figure 7-7. The data are also summarized in Table 7-25. These data have been previously described in detail.

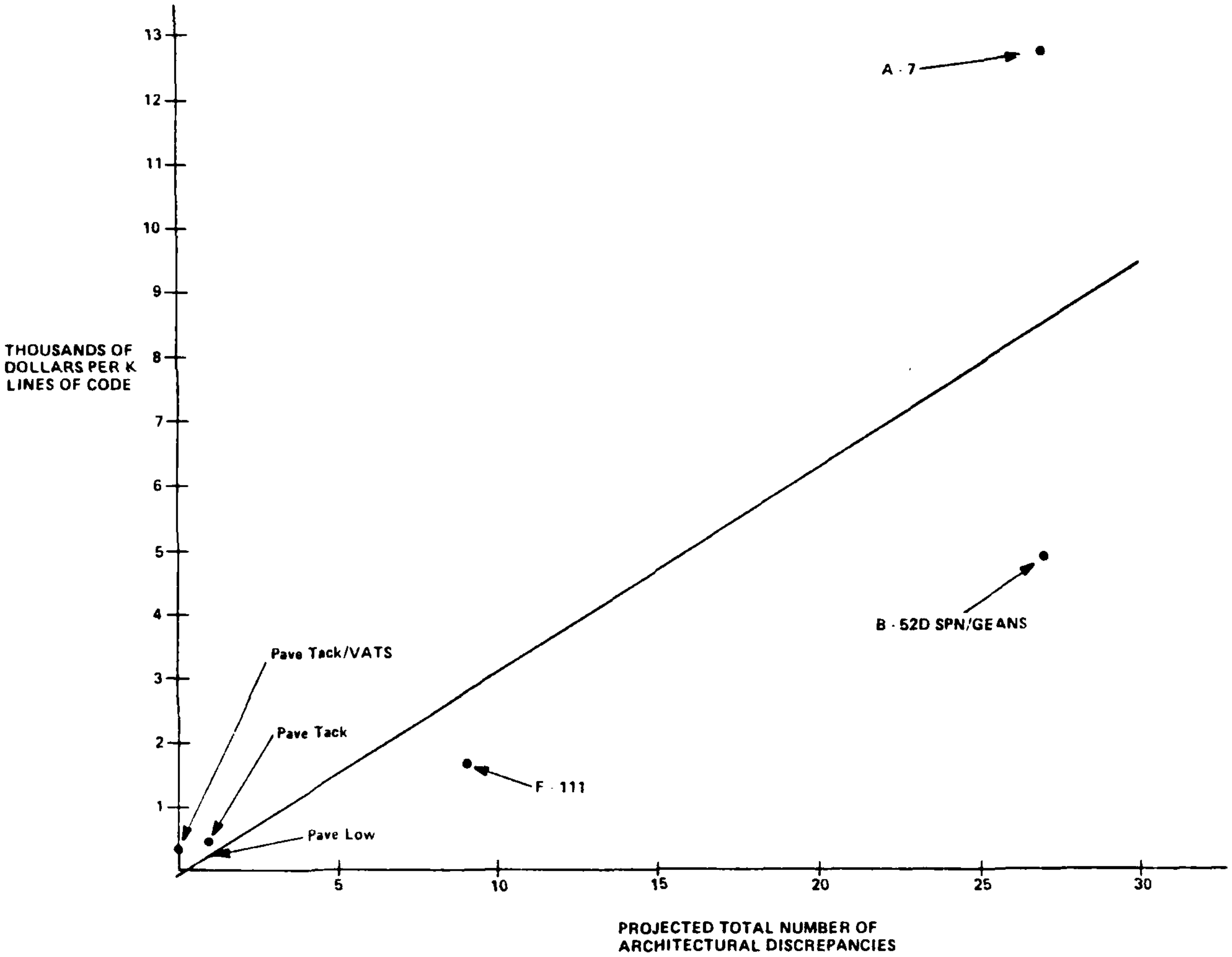


Figure 7-7. Cost versus Architectural Discrepancies

Table 7-25. Summary of Architectural Discrepancy and Revalidation Cost Data

Program	Computer	Total Projected Architectural Discrepancies	Total Revalidation Cost Per K Lines of Code
A-7	TC-2A	29	12,764
PAVE LOW	TC-3B	1	240
PAVE TACK	TC-3	1	481
PAVE TACK/VATS	TC-3A	0	433
F-111	CP-2A	9	1,686
B-52D/SEN GEANS	AP-101C	27	4,882

The best fitting line is also drawn in Figure 7-7. The equation for the line was calculated via linear regression analysis using the equations for slope and intercept described in Appendix E. The equation is:

$$\frac{\text{Dollars}}{\text{K Lines of Code}} = -154 + 320 * \left[\begin{array}{l}\text{Projected Total} \\ \text{Number of Architectural} \\ \text{Discrepancies}\end{array}\right] \qquad (7\text{-}1)$$

The correlation coefficient for this equation is 0.877. This is a measure of the degree of fit of the given points to the least squares straight line. A value of one indicates exact correlation; a value of zero indicates no correlation. In this case, a reasonable fit of the data to the least-squares straight line is indicated.

The negative value of the intercept warrants discussion. If a near perfect verification program were available (i.e., the projected total number of architectural discrepancies after verification would approach zero), the minimum cost to be incurred would be the cost of the basic flight test to validate the operational flight program, or $2,500. To translate this to dollars per K lines of code, this figure could be divided by the average size of the Operational Flight Programs, in this case 15K. This yields a value (which is the expected y-intercept) of $167 per K lines of code.

Naturally, a negative cost for this flight test is physically impossible. However, when the least-squares line of Figure 7-7 is examined, it can be easily seen that a y-intercept of $167 instead of a -$154 would not significantly alter the fit of the line to the data points. In summary, the negative intercept does not severely limit the credibility of the fitted line.

Note that this curve does <u>not</u>, by itself distinguish between different verification methods. But, given the quality (in projected total number of architectural discrepancies) of a particular certification method, the associated cost can be determined.

As an example of the use of this equation, suppose that Verification Method A were determined to have 20 expected architectural discrepancies remaining in a machine the size of a typical MIL-STD-1750 implementation after certification. This would yield

$$\frac{\text{Dollars}}{\text{K Lines of Code}} = -154 + 320 * 20 \text{ (projected architectural discrepancies)}$$

$$= 6,246$$

If one assumed an average Operational Flight Program size of 32K and expected the number of Operational Flight Programs to be written over the useful life of MIL-STD-1750 to be ten, then the total cost is:

$$\text{cost} = \frac{\text{6246 Dollars}}{\text{K Lines of Code}} * \frac{\text{32 K Lines of Code}}{\text{Operational Flight Program}} * \text{10 Operational Flight Programs}$$

$$= \$1,998,720$$

This may be thought of as an estimate of the penalty (cost) that must be paid for having a less-than-perfect verification method (hypothetical Verification Method A, in this example).

7.3.2.4 Discussion of the Data Points for the Cost versus Architectural Discrepancies Relationship

Of the six data points used in constructing the cost versus architectural discrepancies relationship, two have restrictions which reduce the confidence which can be placed in them.

The first is B-52D SPN/GEANS. This program has yet to begin the flight test portion of software validation. Hence, an estimate was made of the expected number of flight tests by a software engineer with significant experience in this area. Naturally, the data point is no better than the estimate. While comparison with the A-7 data point suggests that the estimated number of flight tests may be low and that the cost for B-52D SPN/GEANS may be understated, the data point is used as is with the above caveat.

The second data point in F-111. Both the software validation cost and the projected number of architectural discrepancies are known to be understated. The reason for the cost bias is that one of the programs which was validated was a composite of the two others. As a result, some of the code which was validated was redundant; the cost data is an understatement of the true cost.

The reason for the bias in the projected number of architectural discrepancies lies behind the incomplete nature of the program. IBM submitted an unsolicited proposal to replace the two CP-2 computers onboard the F-111 with a CP-2A computer. A brassboard machine was built and was used for Operational Flight Program revalidation by the Air Force. Only the simulation portion of validation was completed however; no flight tests were performed. Because of the informal nature of the brassboard development, changes were recorded but not formally written up as ECs. Upon completion of the simulation portion of the validation, use of the machine was discontinued. Estimates of the expected number of flight test and associated costs were made by the Air Force, however. In summary, the following deficiencies exist with the F-111 data:

1. The lack of time information does not permit a projection of the total number of architectural discrepancies. Also, the amount of data has been reduced because use of the machine was discontinued. Since the number of ECs is used directly as an estimate of the projected total number of architectural discrepancies, this results in an understatement of the total number of architectural discrepancies.

   Since, in this case, the number of ECs is used directly as an estimate of the projected total number of architectural discrepancies, this results in an understatement of the total number of architectural discrepancies.

2. The flight testing portion of the software validation costs is an estimate.

3. The three OFPs were somewhat redundant in nature, resulting in an understatement of the validation cost per K lines of code.

Since the biases for the F-111 data point are opposing, it is not clear which would dominate any changes to the final relationship shown in Figure 7-7.

In summary, both the F-111 and B-52 SPN/GEANS data points are taken as is with the caveats about credibility above.

7.3.2.5 Analysis of Quality Data for Different Verification Methods

As summarized in a previous section, Step 2 in estimating the costs to be assigned to the different verification approaches requires that, for each verification approach, the number of architectural discrepancies expected to remain after verification be estimated. This would then be applied to the cost versus architectural discrepancies relationship obtained in the previous section.

The data which were collected for Step 2 appeared previously in Tables 7-15, 7-16, and 7-20 and in Section 7.3.1.2. They are summarized in Table 7-26.

Table 7-26. Architectural Discrepancies Data for Different Verification Methods

Verification Method	Computer/Program	ECs Found	Total Projected Architectural Discrepancies
Functional Type	TC-2A/A-7	24	29
	CP-2A/F-111	9	9
	AP-101C/ B-52D SPN/GEANS	26	27
Random Type	System/370 Model	44	68
Lockstep Step	Series/1	0	0

An examination of Table 7-26 indicates that there is insufficient data to satisfactorily discriminate between the three verification methods. While the data for the Functional type certification method appear to be satisfactory, the single data points of the Random type and Lockstep type verification methods do not provide sufficient confidence to permit their application to the cost model. Furthermore, the Series/1 data imply that the Lockstep type method is of perfect quality. This contradicts intuition.

Because of these deficiencies in the data, there is not sufficient confidence to permit distinction between the different verification methods on the basis of quality. The estimation of the costs to be assigned (Step 2) will not be fully completed and the normalization methodology described previously will not be fully applied. This does not completely invalidate the results of the quality investigation, however. Some analysis can be performed with the recognition that only limited confidence can be obtained in the results and the results must be applied with caution. This analysis appears below. The meaning of the inability to complete Step 2, as well as the conclusions to be drawn from the data, are discussed in Section 7.5. A discussion of _a priori_ expectations regarding quality appears in Section 7.4.6.

Even though it is not possible to distinguish between the different verification methods, it is possible to obtain a rough estimate of the cost penalty associated with the use of the Functional approach. Of the three Functional data points in Table 7-26, the F-111 data point is eliminated (for the reasons discussed in the previous section). The projected total number of architectural discrepancies is taken to be the average of the B-52 SPN/GEANS and A-7 data points (i.e., 28). Since these two data points are for machines which are roughly the size of the nominal MIL-STD-1750 machine, and since great precision is not expected for only two data points, normalization by machine size (as described in Appendix B) is not performed.

Applying 28 projected architectural discrepancies to Equation (7-1) yields:

$$\frac{\text{Dollars}}{\text{K Lines of Code}} = -154 + (320 * 28)$$

$$= 8,806$$

This cost per K lines of code is then multiplied by the expected number of lines of operational code to be written over the lifetime of MIL-STD-1750. The Air Force has estimated this to be between 313.2K and 522K lines of operational code. The expected cost range is:

313.2K lines of code * 8,806 dollars/K lines of code = $2,758,039

522K lines of code * 8,806 dollars/K lines of code = $4,596,732

Therefore, the expected cost penalty that would be paid for having a less-than-perfect verification method (in this case, the Functional type) is between $2.8 and $4.6 million.

Further analysis can also be applied to the System/370 model data point. Recall, that this data point is contaminated in that _both_ the Random and Functional approaches were applied. Therefore, one expects that the resulting quality would be higher than that of the Functional approach alone.

This is shown to be the case after the data is normalized by machine size. Table 7-23 shows the distribution of logic-related ECs, microcode-related ECs, and main storage-related ECs. Table 7-26 shows that 44 ECs were found and that the projected total number of architectural discrepancies is 68. The projected number of logic-related architectural discrepancies becomes:

$$\begin{aligned}\text{Projected Logic-Related Discrepancies} &= (68/44) * (26\ \text{logic-related ECs found}) \\ &= 40\end{aligned}$$

The corresponding projected number of microcode-related and main storage-related architectural discrepancies are 22 and 6, respectively.

Normalizing by machine size (using the data from Table 7-26 and the nominal machine size for MIL-STD-1750), and using the method in Appendix B, one obtains:

$$\begin{aligned}\text{Project Number of Architectural Discrepancies} &= 40 * \frac{29,894}{274,000} + \\ &\quad 22 * \frac{611,201}{1,719,296} + \\ &\quad 6 * \frac{1,179,648}{8,388,608} \\ &= 13\end{aligned}$$

This number suggests that the improvements to be obtained by adding the Random approach to the Functional approach would be to decrease the expected number of architectural discrepancies from 28 to 13.

The application of these results appears in Section 7.5.

7.4 COMPARISON

In this section all twelve approaches are compared against the same five guidelines used to evaluate each approach, Non-Recurring Start-Up Costs, Recurring Costs, Time Required to Perform Validation, and Impact to SEAFAC Resources. Tables 7-27 and 7-28 summarize the data described previously. Note that the total size is the summation of the verification program and whatever other control programs or test cases that are developed to support the verification process.

Table 7-27. Comparison Summary

Approach	Total Size	Verification Time	Total Cost Over 10 Years (K Dollars)
AN/AYK-15A ATP			
Manual	96K	48 min	462.4
Master/Slave	104K	11 min[1] 21 min[2]	617
Random			
Manual	162K	86 hrs, 50 min	962.8- 682.8--
Master/Slave	170K	6 hrs, 12 min[1] 7 hrs, 15 min[2]	1133.8- 853.8--
AVP			
Manual	2M	15 hrs, 2 min	1210.4
Master/Slave	2M	13 min[1] 22 min[2]	1397.4
Diagnostic			
Manual	96K	48 min	735.4
Master/Slave	104K	12 min[1] 22 min[2]	890.0
FTP			
Manual	96K	48 min	462.4* 1085.4**
Master/Slave	104K	10 min[1] 11 min[2]	617.0* 1240.0**
Lockstep			
Manual	1.1M	7 hrs, 40 min	847.0 875.0***
Master/Slave	64K+ 64K++ 1.1M	17 min[1] 30 min[2]	1650.1+ 1245.4++ 997.1+++

* Modify Existing FTP

- Write New Simulator

** Write New FTP

-- Modify Existing Simulator

*** With Software Trace

+ With "Golden" Computer

++ With Simulator

+++ With Predetermined Results

[1] Based on MIL-STD-1553 Transfer Rate of 30K words/second

[2] Based on RS 232 Transfer Rate of 3K words/second

Table 7-28. Cost Comparison

Approach	NRSU (K $)	Recurring Cost/Computer (K Dollars)	Total Recurring Cost Over 10 Years (K Dollars)	Total Cost Over 10 Years (K Dollars)
AN/AYK-15A ATP				
Manual	259	6.78	203.4	462.4
Master/Slave	392	7.5	225.0	617.0
Random				
Manual	799- 519--	5.46	163.8	962.8- 682.8--
Master/Slave	952- 672--	6.06	181.8	1133.8- 853.8--
AVP				
Manual	833	12.58	377.4	1210.4
Master/Slave	966	14.38	431.4	1397.4
Diagnostic				
Manual	532	6.78	203.4	735.4
Master/Slave	665	7.5	225.0	890.0
FTP				
Manual	259* 882**	6.78	203.4	462.4* 1085.4**
Master/Slave	392* 1015**	7.5	225.0	617.0* 1240.0**
Lockstep				
Manual	448 476***	13.30	399.0	847.0 875.0***
Master/Slave	1207+ 943++ 602+++	14.77 10.08 13.17	443.1 302.4 395.1	1650.1+ 1245.4++ 997.1+++

NOTES: REFER TO TABLE 7-27.

7.4.1 Test Configurations

A given verification approach can eliminate certain Test Configurations from consideration. A limiting factor surrounding the manual test configuration is the size of the verification program, because under the manual test configuration, program loading and starting requires manual intervention. Therefore, the larger the size of the verification program, the more separate loads it requires. The following verification approaches can be ruled out because of having a greater than practical amount of manual intervention:

Section	Approach
7.3.1	AVP in a Manual Test Configuration
7.3.5	Lockstep in a Manual Test Configuration
7.3.7	Random Instruction in a Manual Test Configuration

This leaves verification programs developed based on FTP, Diagnostic, and AN/AYK-15A approaches which are feasible under a manual test configuration, and which are very similar in design and function. The major advantage (to the vendor) of a verification program running under a Manual Test Configuration is that it easily allows the vendor to "pre-test" the computer before bringing his unit in for certification (the source code for the verification program will be made available to him prior to certification).

All approaches are feasible to operate under the Master/Slave Test Configuration. The disadvantages though is some difficulty (to the vendor) of "pre-testing" when the vendor doesn't have the Master/Slave configuration, including the I/O interface.

7.4.2 Non-Recurring Start-Up Costs

The difference between the lowest Non-Recurring Start-Up cost for developing a verification program (AN/AYK-15A or FTP approaches in a Manual Test Configuration at $259,000) and the most expensive approach (Lockstep in a Master/Slave environment with a "golden" computer at $1,207,000) is $948,000. The key issue surrounding the cost of the FTP approach in a Manual Test Configuration is that the 3.7 man year figure is based on modifying an existing FTP. Currently, there does not exist an FTP for the MIL-STD-1750, but one will exist in the July, 1980 time frame. The AN/AYK-15A ATP currently exists with documentation supporting its design and use.

A second factor to be considered is the line item for MIL-STD-1750 simulators included in the Lockstep and Random Instruction approaches. If an existing MIL-STD-1750 simulator could be installed and modified tc run in conjunction with the verificaticn program, then the 6 man years development figure could be cut to a 2 man year modification cost.

Certain recurring costs are the same for all approaches. They are the bcctstrap and control programs for a Master/Slave Test Configuration, and the source tape generation program (which may be just a system utility for the Manual Test Configuraticn). The cost cf the Master ccmputer, and MIL-STD-1750 support software are considered to be zero.

7.4.3 Recurring Cost

Recurring costs are directly proportional to the time required to ccmplete the verificaticn process. Therefcre, frcm Table 7-27, it is readily observable that only one approach distinguishes itself significantly enough to be ruled unfeasible for implementaticn. The Random Instruction in a Manual Test Ccnfiguration requires 11 work days of manual intervention to accumulate the executicn of 1,250,000 randomly generated test cases.

The recurring cost figures are calculated assuming that no errors are found in the unit under test. If any errcrs are found, this figure wculd increase proportionally. In all the remaining cases, therefore, the recurring figures are all within a one week time frame.

The second ccmponent of recurring cost is the scftware/hardware maintenance cost associated with each approach. The Lcckstep approach with the "golden" computer distinguishes itself as being the only approach which has hardware maintenance cost. Software maintenance ccsts are proportional to the size of the verificaticn program.

7.4.4 Time Required to Perform Validaticn

The time required to perform verificaticn has been defined to be the time between when the verification program is initially loaded into the unit under test, and when it successfully completes execution (no errors found). As can be seen frcm Table 7-27, cnly cne approach's execution time is large enough to immediately eliminate it from further consideration, that is the Random Instruction approach in a Manual Test Configuration. All other apprcaches fall well within a cne week test period time frame.

7.4.5 Impact to SEAFAC Resources

The basic resources that are common to all approaches and test configurations are the following:

- Development Computer
- Support Software
    - MIL-STD-1750 Cross Assembler
    - Linking Loader
    - Simulator
- Development Personnel
- Certification Facility
- Certification Plan Document
- Certification Personnel
- Source Tape Dump Program

The resources which apply to the Master/Slave Test Configuration are:

- Bootstrap Load Program
- Control Program on Master
- I/O Test Program

The Lockstep approach requires the "golden" computer to be purchased, verified and maintained which adds appreciable cost.

7.4.6 Quality Expectations for Different Verification Methods

While the foregoing quality analysis has not resulted in sufficient confidence to discriminate between the verification methods on the basis of quality, some judgements can be made about the expected differences.

First of all, it is likely that there are not significant differences in quality between the Functional approach and the Lockstep approach. Each of these relies on manually generated test cases. Assuming equal ability and insight of the people writing the test cases, there would be no differences in the quality of the test cases.

The two approaches do differ in the method of generating the expected results. In the Functional approach they are generated manually and in the Lockstep approach they are generated by the "golden" machine. Since the expected results are the same in both cases, there should be no impact on quality.

The two approaches may also differ in terms of the checking of architectural entities (e.g., registers, indicators) which should remain unchanged by a test. It is relatively easy for the programmer using the Lockstep approach to check all the status information and all the general registers and all the address registers for any undesired changes. To do this, he needs to keep track of the contents of these registers. To accomplish the same checks using the Functional approach requires that the programmer maintain the expected information. To the degree that the programmer using the Functional method does not check all of the architectural entities for lack of change, the Lockstep method may offer a slightly higher quality. However, the differences would not be expected to be significant.

However, a comparison between the Random approach and either the Lockstep approach or the Functional approach does reveal an expected difference in quality. First, consider a simple hypothesis: the quality of an approach is a function of both the number of test cases and the "value" of those test cases. This appears to agree with intuition. Each additional test case provides some improvement in the quality of the verification method (assuming it is not merely a copy of a previous test case).

Some test cases may be more valuable than others for improving the quality of a verification approach. It would be expected that a collection of one hundred carefully thought-out test cases would be a better check of a machine than one hundred randomly chosen test cases (which may not, for instance, even check all addressing modes or even the ADD instruction). For this reason the Functional type of verification approach would be expected to be of higher quality than the Random approach for the same number of test cases.

The comparison does not end here, however, since the number of test cases must be taken into consideration. Since the test cases for the Functional type of verification approach must be generated manually, the cost of generating each additional test case is significant. A doubling of the number of test cases would result in a near doubling of the cost. For the Random approach, however, in which the test cases are automatically generated, the cost of generating each additional test case is very small. A doubling of the number of test cases would result in only a small increase in total cost.

The result is that it is not clear which approach would provide the greater quality test. The answer will clearly be a function of the duration of the test. Given the opportunity to run long enough, the Random test will surpass the Functional test in terms of quality (since the Functional test is fixed in terms of the number of test cases). The break-even point in terms of time is not known, however.

An anecdote regarding the comparison of the Functional and Random approaches may be of interest. Two different machines were developed by the IBM System Products Division at Poughkeepsie, N.Y. The first machine was developed using a Functional type of test. A Random type test was then applied, and some additional architectural discrepancies were discovered. The second machine was developed using the Random type of test. A Functional test was then applied, and no additional architectural discrepancies were found. This suggests that the Random test is of higher quality.

7.5 INTERPRETATION OF COST MODEL RESULTS

Since the quality analysis was unable to provide a credible cost penalty figure (due to less-than-perfect quality) for each of the verification approaches, these costs cannot be considered in the cost model. This does not, however, reduce either the reality or the significance of these costs. Neither is the quality analysis completed invalidated.

Ignoring, for the moment, the cost penalty of having a less than perfect verification approach, one is left with the conclusion that the lowest cost verification method (as shown in Table 7-28) is the best one. This is simply because there is no firm data to differentiate between the methods in terms of quality. The modification of the AN/AYK-15A Acceptance Test Procedure (ATP) is the lowest cost and, therefore, the best method (in the absence of quality considerations). The previous comments about the availability of an FTP eliminate the FTP approach, which has the same cost of modification of the AN/AYK-15A ATP approach.

The quality analysis does provide valuable insight into the cost penalties associated with this method, however.

The AN/AYK-15A ATP is of the Functional generic type, as discussed previously. As described in Section 7.3.2.5 sufficient data do exist to provide a rough cost estimate (due to less than perfect quality) of the Functional type of verification method.

The results of that section indicate that the expected cost penalty that would be paid for having a less-than-perfect verification method (in this case, the Functional type) is between $2.8 and $4.6 million.

This is a significant cost. Obviously, if it were possible to significantly improve the quality of the overall test at a cost significantly less than the $2.8 to $4.6 million range, this would be very desirable. The data gathered strongly suggest that the Random approach to architectural verification is the best method of augmenting the quality of the AN/AYK-15A ATP approach.

Three reasons apply. First, the Random approach provides an extremely large number of test cases, many times the number of test cases of the AN/AYK-15A ATP approach. Each new test case provides a positive increment to the overall quality.

Second, the test cases generated under the Random approach are expected to be independent of (i.e., not related to) the test cases used in the AN/AYK-15A ATP approach. This is simply because in the Random approach they are generated randomly and in the AN/AYK-15A ATP approach they are generated manually. The other verification methods examined in this study also have manually generated test cases. They would, therefore, be expected to be less independent. This independence is required in order to obtain improvements in quality.

Third, the Random approach provides, by far, the largest number of test cases per dollar expended.

The two methods added together should, therefore, improve the total quality. The anecdote in the previous section implies that the Random approach offers a higher quality. Furthermore, testing at SEAFAC should permit a Random test of long duration, since overnight and weekend testing is possible. About 28.8 million test cases should be possible in a week long verification test.

The cost of implementing the Random certification method, as shown in Table 7-28, is well below the $2.8 to $4.6 million range discussed above. If the overall quality of the test were improved by 22 percent by the addition of the Random approach, then the reduction of the cost penalty by $622,000 = ($1,239,000 - 617,000) (22 percent of $2.8 million) would offset the cost of implementing the Random verification method. It seems reasonable to expect that the addition of 28.8 million test cases to the approximately five thousand test cases in the AN/AYK-15A ATP method would result in a quality improvement of 22 percent. ($1,239,000 is the total cost for the IBM recommendation which will be discussed later.) While the data gathered in this study cannot definitively prove it to be the case, it appears that investment in the combination Functional/Random approach will be easily recovered.

As was mentioned previously, caution must be used when drawing conclusions based on the $2.8 million to $4.6 million cost penalty. However, there is good reason to believe that any errors will be on the side of conservatism. This is because the actual cost penalty to the Air Force is likely to be substantially higher than the $2.8 million stated. If this is the case, then the justification for adding the Random approach is further reinforced.

Consider again the quality analysis. The cost penalties measured were for the correction of architectural discrepancies discovered during Operational Flight Program validation. The costs here are limited by the fact that, since this is still in the development phase, changes only need to be made to the development hardware or software, not to many machines in the field. But, just as some architectural discrepancies escape detection during architecture verification, some architectural discrepancies will escape detection during software validation. Experience shows that, even after years of operational use, additional architectural discrepancies can be discovered. This could be for many reasons; previously untried data combinations are certainly one possible cause.

Correction of architectural discrepancies which are discovered after operational use of the hardware and software has begun can cost several order of magnitude more than correction of those same architectural discrepancies caught during the development phase of a program. This is simply due to the tremendous logistics involved in changing many computers which are in the field. Experience shows that these costs could easily overshadow the $2.8 to $4.6 million estimate made above. This further emphasizes the value of improving the

AD-A099 260 IBM FEDERAL SYSTEMS DIV OWEGO N Y F/G 9/2
MIL-STD-1750 CERTIFICATION STUDY.(U)
FEB 80 M L KUSHNER, D C REISIGER, W J TRACZ F33657-79-M-0858
UNCLASSIFIED IBM-6176175A NL

3 OF 3
AD
A099260

END
DATE
FILMED
[illegible]
DTIC

quality of the verification test to reduce the large cost penalties which result from having less-then-perfect verification.

Naturally, the question may be asked: If the Random approach is of higher quality, why not eliminate the Functional approach? The answer is two-fold. First, the combination of the approaches provides an even higher quality. Second, and perhaps the more important to the Air Force, the Functional approach provides a certain minimum test of the machine. The Random approach, while providing perhaps a higher degree of quality of test, cannot provide any minimum degree of testing. For example, it is theoretically possible that, after a million test cases, that the ADD instruction would remain untested. While this is highly unlikely, the ability to provide a specific level of testing must be of considerable importance to the Air Force.

Another question may be asked as to why this combination Functional/Random approach has not been applied to avionics machines in the past. The answer is based on simple economics: for single applications of a computer architecture, the test investment must be recovered in that single application; multiple use architectures like MIL-STD-1750 can recover the test investment over those multiple applications. Effectively, there are economies of scale in applying the test over and over again.

The quality analysis in this report provides some benefits to the Air Force in addition to helping select the best verification approach. It should assist the Air Force in developing realistic expectations regarding the certification process and the use of MIL-STD-1750 machines. The collected data show that the verification method will not be perfect. Machines will be certified that have hidden architectural discrepancies. Previously validated software will not always execute correctly on newly certified machines.

For the users of MIL-STD-1750 machines, it provides a conceptual framework for the error discovery process during Operational Flight Program validation. Understanding of these results should permit more accurate budgeting and scheduling for programs that use MIL-STD-1750 computers.

In summary, the cost model has provided valuable data for discriminating between the different verification approaches. The quality analysis, while unable to provide high confidence in the specific dollar penalties to be assessed each verification approach, has provided an estimate of the penalty to be paid for the use of a single verification test (in this case Functional) approach. This information, combined with the cost data, provide the basis for the recommendation of a combined Functional/Random approach.

This recommendation will be discussed in detail in the following section.

THIS PAGE INTENTICNALLY LEFT BLANK

8.0 RECOMMENDATION

This section contains IBM's recommendation to the Air Force for implementing the MIL-STD-1750 certification capability in SEAFAC. It consists of a description of the approach, and the rationale leading to its selection. A summary of the resources required is also covered. As a conclusion, an outline for the MIL-STD-1750 Certification Test Plan which describes the chain of events leading up to vendor certification is presented.

8.1 TWO LEVEL APPROACH

8.1.1 Description

IBM recommends a two phase approach to certification as follows:

Phase 1 A deterministic verification approach based on modifications to the AN/AYK-15A ATP is run on the MIL-STD-1750 computer being tested. (These modifications delete the non-MIL-STD-1750 features from this ATP.) This provides a predefined minimum test of the machine using 5,000 test cases. This first phase tests the integrity of each MIL-STD-1750 instruction for selected data patterns and addressing modes as described in the Military Standard Airborne Computer Instruction Set Architecture document and tests the boundary problem areas in the MIL-STD-1750 architecture.

Phase 2 After completion of Phase 1, a Random verification approach is used to generate large numbers of test cases. The second phase of testing verifies that large number of instruction sequences execute according to their intended function. A test case contains an instruction which is first run on a MIL-STD-1750 simulator and then run on the MIL-STD-1750 computer being tested. The simulator results are compared with the results from the MIL-STD-1750 computer being tested. 28,800,000 test cases should be possible during a week long verification test.

Both phases of testing would be conducted under a Master/Slave Test Configuration. The MIL-STD-1553 Serial Channel I/O interface is recommended as being the standard to which each vendor should target his certification configuration; but a multiple interface capability is recommended -- MIL-STD-1553 or RS-232. Ramifications of using a

RS-232 Serial Channel I/O interface are identified in Section 7.

This provides a superior means to accomplish the certification process because:

1. The 2-phase process (with Random as the second phase) provides a higher quality than any single verification approach.

2. The 2-phase process provides a minimum testing level.

The recommended approach for arriving at a verification program that fulfills the requirements of the first phase of testing, is accomplished by the modification of the AN/AYK-15A ATP. The recommended approach for arriving at a verification program that satisfies the requirements of the second phase of testing is the development of a Random Instruction Generator Program, and the modification of the existing MIL-STD-1750 simulator to interface with a control program that compares the results.

The verification testing can be achieved in less than two weeks of time.

8.1.2 Rationale

The rationale for selection of a two level approach to architectural verification was discussed in detail in Section 7. This rationale is summarized here.

The cost model shows that the lowest cost approach is the modification of the existing AN/AYK-15A Acceptance Test Program. This method, then, constitutes the first level of verification for MIL-STD-1750 machines.

The quality analysis shows that use of this single method results in a cost penalty to the Air Force of between $2.8 and $4.6 million. This penalty is the additional software validation costs that is incurred over the lifetime of MIL-STD-1750 because of the less-than-perfect quality of the AN/AYK-15A ATP. The magnitude of this cost suggests that use of an additional method of verification testing to improve the overall quality could substantially reduce the overall cost and at the same time provide SEAFAC with a better test.

The data gathered strongly suggest that the Random approach to architectural verification is the best method of augmenting the quality of the AN/AYK-15A ATP. Three reasons apply. First, the Random approach provides an extremely large number of test cases, many times the number of test cases of the AN/AYK-15A ATP. Each new test case provides a positive increment to the overall quality.

Second, the test cases generated under the Random approach are expected to be independent of (i.e., not related to) the test cases used in the AN/AYK-15A ATP. This is simply because in the Random approach they are generated randomly and in the AN/AYK-15A ATP approach they are generated manually. The other verification methods examined in this study also have manually generated test cases. They would, therefore, be expected to be less independent. This independence is required in order to obtain improvements in quality.

Third, the Random approach provides, by far, the largest number of test cases per dollar. It is reasonable to expect that the additional cost of the Random approach ($622,000) is offset by a reduction in the cost penalty due to the improvement in quality. This requires an improvement in quality of 22 percent. It seems reasonable to expect that increasing the number of test cases from 5,000 to 28,800,000 would provide, at least, a 22 percent improvement in quality, although there is no firm data to support this.

Comments about the two level approach with regards to the evaluation criteria follow.

QUALITY

The two level approach affords a higher degree of quality than any single approach. The two levels are complementary in that the AN/AYK-15A ATP approach provides a predetermined minimum test of all instructions (but not a large number of test cases) and the Random approach provides a large number of test cases (but does not provide any certainty for testing all instructions).

The test cases from Random are expected to be independent from the test cases generated by the ATP. That is simply because in Random they are generated randomly and in the AN/AYK-15A ATP they are generated manually. This independence is required for an improvement in quality.

The quality of the certification approach can be increased by allowing the Random instruction verification program to run for longer periods of time. Under the Master/Slave Test Configuration, the Random instruction verification program can be left to run unattended during off hours (on second and third shifts or over the weekend).

COST

The Non-Recurring Start-Up cost for implementation of the IBM recommendation within a year and a half is $933,000; the Recurring cost for 30 computers over 10 years is $306,000; the total cost is thus $1,239,000 for the implementation of the recommendation.

The AN/AYK-15A ATP approach was shown by the cost model to be the lowest cost approach. Addition of the Random approach will increase the cost to SEAFAC, but should result in an overall cost reduction to the Air Force because of the increase in quality.

TEST CONFIGURATICN

The Master/Slave Test Configuration is recommended. The Master/Slave Test Configuration provides the only feasible environment in which the Random Instruction Verification approach exists. However, the AN/AYK-15A ATP approach runs under either a Manual or Master/Slave Test Configuration.

VENDOR IMPACT

The AN/AYK-15A ATP approach will yield a verificaticn program of which each vendor could utilize portions to pretest his MIL-STD-1750 prior to certification provided that the scurce program be made available to the vendor. This availability will decrease the likelihood of finding "first order" errors in the unit under test. Additionally, the Random verification approach's Generator Program and Simulatcr Program could also be used by the vendor to further pretest his MIL-STD-1750 computer prior to the certification process in order to increase confidence in passing the certificaticn process.

IMPACT TO SEAFAC

SEAFAC's VAX 11/780 can be used as the Master computer for the certification process. Suppcrt software would be available on it for develcpmental and maintenance activities. SEAFAC personnel (an observer, a technician and a coordinator) is needed during the certification process. One programmer is needed tc sustain the Acceptance Test Program and the Random prcgrams; alsc, five software programmers are required to develop the Random program (within one year) unless this is subcontracted outside of SEAFAC. If the major software components resident on the Master Computer (Simulator, Random Instruction Generator) are written in an HCL, the verdor is able to install them on a computer system other than the VAX 117/80, thus facilitating pretesting at the vendor's facility.

VERIFICATICN TEST TIME

The certification process takes less than two weeks for a 500 KOP computer using a MIL-STD-1553 channel tc a VAX 11/780. (It takes 7 full days; but two weeks should be allocated for the certification process.) If a 150 KOP computer (not a 500 KOP computer) is submitted, the time for certification expands less than 10 percent (frcm 7 to 7.7 days), but remains less than two weeks. If a 500 KOP computer is submitted with an RS-232 interface, the time for the certification process expands 17 percent (frcm 7 full days tc 8.2 full days) if concurrent overlapped CPU and I/O processing is not permitted in the VAX 11/780; but remains less than two weeks. If concurrent overlapped CPU and I/O processing is permitted, the CPU processing still dominates and completed hides the I/O transfer. In this case, the RS-232 channel would have nc discernable impact to the time required for the certificaticn process.

TEST I/O INTERFACE

With the MIL-STD-1553 Serial Channel interface planned for the VAX 11/780, it should be used in the Master computer role during the conduction of the certification process to connect to the computer being tested. (Since the MIL-STD-1750 Serial I/O Channel already exists on the PDP 11/55, the PDP 11/55 could bridge the time until the VAX 11/780 system with its MIL-STD-1553 I/O channel is available.) The MIL-STD-1553 Serial Channel requirement is considered to have the least impact on the vendor's hardware configuration since it is expected that from 90 to 95 percent of of the computers being submitted for certification will contain a MIL-STD-1553 channel. However, an RS-232 serial channel is also recommended for those computers without the MIL-STD-1553 channel.

Parenthetically, any parallel channel with a transfer rate of approximately 1 mega words per second is rejected because it speeds up verification about 2 percent from 7 to 6.87 days and is not a significant improvement. That is, the verification process is not I/O bound, but is CPU bound by the VAX 11/780.

8.2 IMPLEMENTATION CONSIDERATIONS

The following section outlines the major items surrounding the implementation of the two level certification process.

8.2.1 Phase 1 - AN/AYK-15A ATP Modification

This Acceptance Test Plan (ATP) for the AN/AYK-15A consists of the following subtests:

- User's Console
- Instruction Set
- Register
- Main Storage
- Bus Controller
- Input/Output
- Power ON/OFF Sequencing

An ATP resident on the AN/AYK-15A is executed according to procedures described in Air Force document, PA 401 207, in which a user interacts with the console keyboard. Each subtest may require a separate processor load. A standalone test is one in which only the processor and an attached console (keyboard and CRT display) are required in order to run the ATP. The Bus Controller Test is not a standalone test in that it requires a configuration which includes the processor under test and at least one other Master/Remote device interfaced via a multiplex bus. Also, various tests require use of external test equipment (e.g., logic analyzers, oscillcscopes, etc.) and/or special purpose Input/Output exercisers.

In order to develop a verification program to fulfill the requirements of the first phase of testing, modifications to both the control program and subtests of the AN/AYK-15A Acceptance Test Plan are necessary. These modifications are summarized in Table 8-1.

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
Keyboard/CRT	Test all keyboard/CRT keystrokes. Print all keystrokes on CRT (keyboard and CRT in Local mode).	Deleted for MIL-STD-1750 Verification
User Command	Verify correct execution of all user commands. Exercise all defined user commands.	Deleted for MIL-STD-1750 Verification
Floppy Disc/ Printer	Verify proper operation; response to user commands. "Wrap" processor main storage from floppy disk to printer via user commands.	Deleted for MIL-STD-1750 Verification
Basic Instruction (done for each instruction)	Test Basic Instruction Operation. Exercise Instruction with known input parameters and compare with expected results.	Modify for MIL-STD-1750 Verification as necessary
Arithmetic Instruction Test	Test all arithmetic instructions and status. Exercise all arithmetic instructions using selected values in the 4 quadrants; check condition status setting.	Modify for MIL-STD-1750 Verification as necessary
Condition Status (done for each application)	Verify setting of the status word for each instruction. Cause result of exercised instruction to set all required combinations of status bits.	Modify for MIL-STD-1750 Verification as necessary

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
Indexed Addressing (done for each applicable instruction)	Test those instructions using index capability. Use 15 index registers; positive and negative indexing.	Modify for MIL-STD-1750 Verification as necessary
Overflow/ Underflow (done for each instruction)	Test those instructions which cause underflow/ overflow; verify underflow/overflow interrupts. Induce underflow/overflow resultant.	Modify for MIL-STD-1750 Verification as necessary
Benchmark	Measure processor throughput. Use benchmark arithmetic and logic instructions mixes and measure average processor throughput.	Deleted for MIL-STD-1750 Verification
Bang Test	Check for illegal sequences of instructions. Use random sequence of instructions and test for invalid indicators.	Deleted for MIL-STD-1750 Verification
Illegal Instruction	Check fault register setting, machine error interrupt, clear fault register output command. Execute illegal instruction and test for expected response.	Modify for MIL-STD-1750 Verification as necessary
General Register	Testability to address and set/reset all 16 general registers. Wrap known patterns from processor main storage.	Modify for MIL-STD-1750 Verification as necessary

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
ECM Control Registers	Testability to address and set/reset all 16 ECM control registers; verify and reset the test bit, compare register and store register input and output commands. Wrap known data patterns from the processor general registers via input and output instructions.	Deleted for MIL-STD-1750 Verification
ECM General Registers	Testability to address and set/reset all 16 ECM general registers. Wrap known data patterns from the processor general registers via input and output instructions.	Deleted for MIL-STD-1750 Verification
Status Register	Testability to set/reset the status register. Wrap known data patterns from general registers via input/output instructions.	Modify for MIL-STD-1750 Verification as necessary
Instruction Counter	Testability to sequence instruction counter through 64K main storage. Increment counter and verify execution of every increment instruction.	Modify for MIL-STD-1750 Verification as necessary
Interrupt Mask	Testability to set/reset interrupt mask register. Wrap known data patterns from general registers via input/output instructions.	Modify for MIL-STD-1750 Verification as necessary

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
Write Protect RAM	Testability to set/reset write protect RAM. Wrap known data patterns from general registers via input/output instructions.	Modify for MIL-STD-1750 Verification as necessary
Main Storage Integrity	Testability to address, write and read 64K main storage. Wrap known worst case data patterns from general register (1's, 0's, and addresses).	Modify for MIL-STD-1750 Verification as necessary
Processor CPU Write Protect	Testability to protect main storage in increments of 1K blocks. Test main storage protect enable output command; test fault register setting. Store data into protected areas and compare for expected results.	Modify for MIL-STD-1750 Verification as necessary
Read Only Memory (ROM)	Test ROM integrity; test enable and disable output command discretes. Perform cyclic checksum within ROM memory using ROM enable/disable output instructions.	Modify for MIL-STD-1750 Verification as necessary Optional in MIL-STD-1750
Illegal Main Storage Address	Verify setting of fault register indicator. Remove main storage module and attempt to read/write vacant locations.	Modify for MIL-STD-1750 Verification as necessary
Main Storage Access	Test for main storage access and cycle time. Instrument main storage controller and measure access and cycle times.	Deleted for MIL-STD-1750 Verification

Table 8-1. MIL-STD-1750 Applicability cf ATP Subtests (ccnt)

AN/AYK-15A Subtest	Purpose	MIL-STC-1750 Applicability
Main Storage Access Priority	Verify priority of Eus Contrcl Module (BCM), Direct Memory Access (DMA) and CPU access to main storage. Initiate simultaneous main storage requests and instrument main storage controller.	Deleted for MIL-STD-1750 Verification
ECM Internal Commands	Test NOOP, LINK and HALT BCM instructions; BCI Level 1 interrupt. Exercise instructions in predefined order (standalone only).	Deleted for MIL-STD-1750 Verification
ECM Self-Test	Test capability cf ECM and MTU to wrap self-test pattern, test bus activity discretes. Exercise BCM self-test function (standalone only).	Deleted for MIL-STD-1750 Verification
Undefined Mode Ccmmands	Test Master/Remcte response to undefined mode commands; ECI Level 2 interrupt. Exercise predetermined mode command message sequence (requires Master and Remote).	Deleted for MIL-STD-1750 Verification
MTU Shutdcwn Mode Commands	Test Master/Remote response to MTU shutdcwn mode commands. Exercise predetermined mode ccmmand message sequence (requires Master and Remote).	Deleted fcr MIL-STD-1750 Verification

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
Mode Commands with Interrupts	Test Master/Remote response to those mode commands which generate remote interrupt. Exercise predetermined mode command message sequences (requires master and remote).	Deleted for MIL-STD-1750 Verification
Synchronous Data Transfers	Test Master/Remote data addressing, tag word storage, data storage. Exercise predetermined message sequences, rotating word count and subaddress.	Deleted for MIL-STD-1750 Verification
Asynchronous Data Transfers	Test Master/Remote ASYNC data addressing, ASYNC message protocol. Exercise predetermined message sequences, rotating word count and subaddress.	Deleted for MIL-STD-1750 Verification
Timer	Test capability to load, start and stop timers A and B. Test capability to input timers A and B. Test timer A and B interrupts. Output various timer values and test using known instruction execution times.	Modify for MIL-STD-1750 Verification as necessary
External Discretes	Test capability to set, reset and read 8 output/input discretes; Test Read Discrete Output and input/output buffer input commands. Wrap discrete output to input, output fixed data patterns and compare.	Modify for MIL-STD-1750 Verification as necessary

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
PIO	Test capability to execute PIO. Output to PIO and compare to output buffer read; use external test hardware to provide external wrap.	Modify for MIL-STD-1750 Verification as necessary Optional in MIL-STD-1750
Interrupts	Test capability to respond to 6 external interrupts, test the clear, disable, enable, and clear pending interrupts output commands. Use external test hardware to initiate interrupts.	Modify for MIL-STD-1750 Verification as necessary Optional in MIL-STD-1750
DMA	Test DMA interface and enable/disable DMA output commands. Test DMA write protect and the fault register indicator Bit 1. Use external test hardware to initiate DMA read/write.	Modify for MIL-STD-1750 Verification as necessary
Trigger GO Indicator	Test for output discrete and 40 ms timeout. Use oscilloscope to measure timeout.	Modify for MIL-STD-1750 Verification as necessary
Illegal	Check fault register setting. Execute illegal output command and test for expected results.	Modify for MIL-STD-1750 Verification as necessary

Table 8-1. MIL-STD-1750 Applicability of ATP Subtests (cont)

AN/AYK-15A Subtest	Purpose	MIL-STD-1750 Applicability
Power Off	Verify power-down interrupt; 100 microsecond hold time. Initiate power down; execute consecutive memory stored after power on.	Modify for MIL-STD-1750 Verification as necessary
Power On	Verify power-on discrete; execution from bootstrap ROM; power-on reset discrete, power-on BCM quiescent state. Initiate power-on; execute from ROM to read/reset power-on discrete; verify BCM state by reading control registers.	Modify for MIL-STD-1750 Verification as necessary Optional in MIL-STD-1750

8.2.2 Phase 2 - Random Instruction Generator Development

To complete the second phase of testing, a Random Instruction Generator Program, and related control programs must be developed, and the existing MIL-STD-1750 simulator modified. The following is a list of characteristics of this approach:

1. The Random Instruction Generator must be designed to accept, as user input, a seed, from which the random sequence of instructions and data can be generated.

2. The MIL-STD-1750 simulator must be modified to facilitate the inspection and modification of all registers and core locations as well as start and stop control. It must be hosted on the VAX 11/780.

3. The following describes this Random process.

8.2.2.1 Random Program Description

It is possible to split the overall program into separate structures or routines. This is done in order to develop detailed operations based upon individual module function. The structure is depicted (Figure 8-1) with names associated with individual functions. This reflects the overall concept only, since further detailing obviously requires a further refinement of the functions.

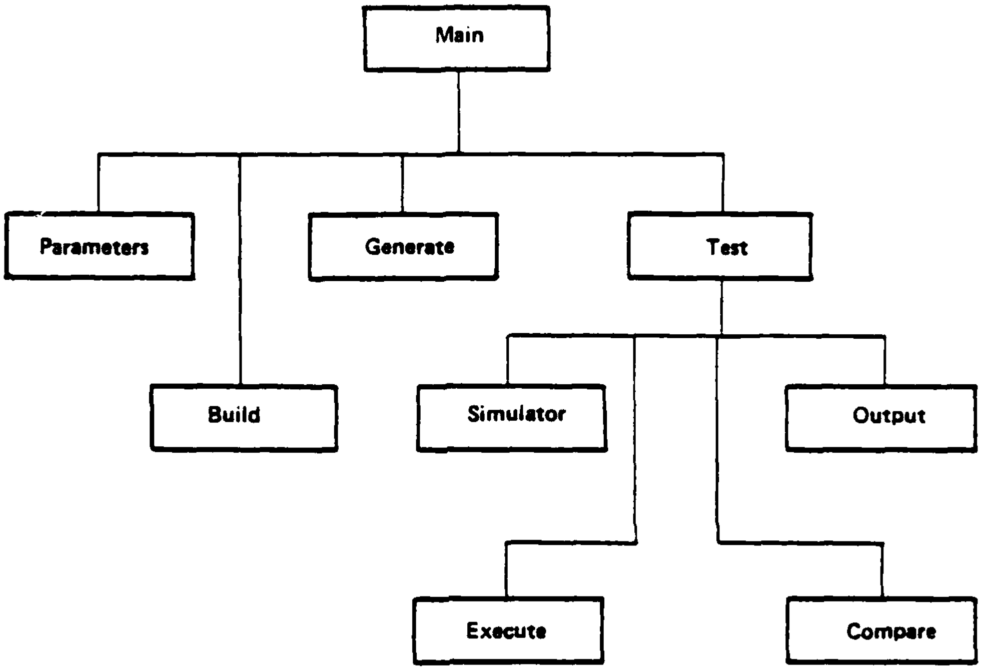


Figure 8-1. Program Hierarchy

8.2.2.1.1 <u>Program Description</u>. In reference to Figure 8-1, a brief description of each block can provide a better understanding of the overall concept:

1. "MAIN" controls the program cycling; the generation, simulation and execution of test cases.

2. The "PARAMETERS" subroutine provides parameter values for use by the remainder of the test program.

3. The "BUILD" subroutine creates the operation code pool from which OP CODES are taken for the pseudo-random instruction stream.

4. The "GENERATE" subroutine builds the instruction stream by randomly selecting an OP CODE and then building the operands for each instruction.

5. The "TEST" subroutine controls test case simulation and execution until an error is detected or the end of the instruction stream is encountered.

6. The "SIMULATOR" subroutine simulates each instruction found within the pseudo-randcm instruction stream.

7. The "EXECUTE" subroutine causes the execution of the instruction stream by the processor.

8. The "COMPARE" subroutine compares the expected results from the Simulator subroutine with the actual results from the Execute subroutine to determine if an error has been generated.

9. The "OUTPUT" subroutine produces all hardcopy resulting from test program execution.

8.2.2.1.2 "<u>MAIN</u>" <u>Routine</u>. The "Main" routine is the topmost module in the series which comprises the Randcm Instruction Test Program. Its function is to control the executicn of the program. The following operations take place within this routine.

- A Program Control Area (PCA) provides the area for parameter input from the operator and other system supplied values.

- Restart is checked by a test of the restart bit in the PCA. If restart has occurred, control is passed to Output routine to provide the operator with restart information.

- The machine architecture is determined by testing the appropriate field in the PCA. Once the architecture is determined, the instruction table for that architecture is brought into the system, if not currently loaded as in the case of restarts.

- A storage area is maintained for module communications. This communications control block contains pointers, structures, and flags for usage by other modules.

- An initial machine state is established.

- The "MAIN" program loops, generating test cases until an operator initiated stop is encountered.

Control is passed in the following sequence to subroutines which make up the "MAIN" program locp.

- The "PARAMETERS" routine: Provides parameter values for the remainder of the test program.

- The "BUILD" routine: Generates an OP CODE pool for use in instruction stream generation.

- The "GENERATE" routine: Generates a pseudo-random instruction stream.

- The "TEST" routine: This routine causes all the testing by passing control to:

    - The "SIMULATOR" routine: Simulates the instruction stream.

    - The "EXECUTE" routine: Executes the instruction stream.

    - The "COMPARE" routine: Determines if an error has occurred.

    - The "OUTPUT" routine: Provides hardcopy output.

8.2.2.1.3 "PARAMETERS" Routine. Parameters, after being validity checked by the parameter processor of the "MAIN" program, become available to the "MAIN" program when a request is made for operator input. The function of this routine is to provide parameter values for use by the remainder of the "MAIN" program. Those values may be a randomized selection of the operator input or a randomized selection of the available options. Operator input is verified and a randomizing of values is performed for each pass through the "MAIN" program. The operation of this routine follows:

- A base seed for random generation of data is requested. Note that loading this base seed permits a computer that has previously failed certification to be retested using the same psuedo-random test sequence.

- A check is made to determine if operator input is available.

- If input is supplied, the buffers size parameter (which is used to track storage allocation and usage) is checked for values within specified ranges.

  If not in range, a request is reissued with a message indicating the operator error.

- If buffer size differs from its previous value, then storage is released and the new storage size is used.

- OP and NOP parameters which control execution are checked for valid OP CODES, and if the same OP CODE is found in both parameters, the NOP parameter overrides the OP value. The

OP and NOP parameters are used to define which of the instructions have been implemented in the computer under test.

8.2.2.1.4 "BUILD" Routine. The function of this routine is to provide a table of valid OP CODES for use by the remainder of the program. By generating a separate table, in lieu of the architecture table, there is a reduction in the amount of time spent in verification of OP CODES found in the random instructicn stream. This CP CODE pocl is formulated in the fcllcwing manner.

- On entry to the routine, pointers are available to the OP parameter OP CODES, NOP parameter CP CCDES, the architecture table, and to the storage area which contains the OP CODE pool.
- Depending on the contents of the OP or NOP parameters, all OP CODES for the randomly selected machine features are processed.
- If OP is specified, only those CP CODES are transferred to the OP CODE pocl.
- If NOP is specified, all OP CCDES with the exception of those specified are transferred tc the OP COCE pool.
- This routine loops until all CP CODES are processed.

8.2.2.1.5 "GENERATE" Routine. The Generate routine is responsible fcr providing a random instruction stream with the appropriate values established in the corresponding contrcl registers, general purpose registers, floating point registers and data areas. In order that the test program is able to distinguish between processor execution results and simulation results, twc copies of the randomly generated instruction stream are available. Therefore, one stream is used as a reference, one stream for simulation and one stream for processor execution. The function of this module is accomplished by:

- A random selection of CP CCDES from the OP CCDE pocl previously built.
- As each OP CODE is placed intc the randcm stream, the operands for that particular CP CODE are formed.
- As operands are placed into the random stream, any register or data area associated with the operand is provided.
- Interrupts are forced into the stream.
- Two copies of the randcm stream and data areas are made available.

8.2.2.1.6 "TEST" Routine. The function of this routine is to continue test case execution until either an ending sequence for the instruction stream or an error is detected. This routine controls the sequence of execution of the "TEST", "EXECUTE", "CCMPARE" and "OUTPUT" rcutines. The sequence of events through this module is:

- Preparation of pointers followed by the simulation of the random instruction stream.
- Provide a randcm stream for execution on the processor and the saving of the results cf that execution.
- Cause a comparison of the results of the simulation to the results of the execution tc determine the extent of success of processor execution.
- Cause a hardcopy of stream and/or data areas based on operator parameter selecticn.
- Cause an attempt to isclate cn a detected processor execution error.

8.2.2.1.7 "SIMULATOR" Routine. This routine simulates the instruction provided to it from the randcm instructicn stream. This is accomplished in two basic steps: (1) a simulation of instruction fetch as performed by the processor unit, and (2) a simulation of the instruction utilizing a basic set of instructicns for simplicity. A frrther breakdown of events fcllows.

- The instruction length code and address of the next sequential instruction are calculated and saved.
- The architecture table entry for the OP CODE of this instruction is obtained to determine the various types of interrupts which can occur from this instructicn.
- A determinaticn is made as tc whether this instruction is capable of causing interrupt to occur.
- The OP CODE is checked tc determine if it is legal, privileged or illegal with corresponding interrupt if necessary.
- The OP CODE format is used to determine if cther interrupts can occur from the specificaticn cf the operands.
- If no interrupt is detected tc this point, the instruction is simulated.

8.2.2.1.8 "EXECUTE" Routine. This rcutine provides the randomly generated instruction stream to the processor for execution. The system environment is maintained by trapping the state of the machine prior to instruction stream execution and restoring that state upon return from execution. The events which cause this transition are:

- The current state of the machine (that is; the contents of general purpose, floating point and control registers as well as the system status) is obtained and saved.
- The register contents required for stream execution replace the general purpose and floating point registers.
- Any control registers are altered tc reflect the requirements of the stream to be executed.
- The Machine State (Instruction Counter, Status Word, and System Interrupt Mask) to cause the random instruction stream to start execution on the processor is established.
- An "in-the-stream" indicator is set to identify the last location of execution.
- The new state of the system is imposed and the processor begins to execute the random instruction stream.

Control does not return until the execution of the random stream results in an interrupt and that interrupt is processed. At this time, the "in-the-stream" indicator is reset and the Exerciser routine is complete.

8.2.2.1.9 "COMPARE" Routine. This routine compares the results of the Simulator routine from Simulating instructions and the results of the Execute routine from processor execution to determine if an error has occurred.

In general, all machine state and data areas (including registers) from the Simulator and EXECUTE routines should match at the time this routine receives control. There may exist cases which are special and which would not result in equality between all fields; they are architectural dependent and must be defined "a priori", and handled specially.

8.2.2.1.10 "OUTPUT" Routine. Hardcopy results of the test cases are provided via this subroutine. A varying amount of printer output is available based on operator specified parameters, values of operands within the random instruction stream, and the location of an error if one has been detected.

On CPU restart, the following information is provided:

- A display of the program control area.

- A display of the operator selected and program randomized parameters.
- A display of the OP CODE counters, i.e., instructions which have been executed.
- If a random instruction stream exists, then a display on:
    - The random instruction stream in a readable format.
    - The interrupt history, or the results of the acutal instruction stream execution versus the results of the simulation of the instruction stream.
    - The control, general purpose, and floating point registers associated with the stream.
    - Any data area which is referenced by the stream.

Error information and/or trace information is provided dependent on the operator selection of a "PRINT" parameter. If an unexpected interrupt or a data error occurred and a printout is desired then the following information is provided:

- The program history, if requested via a "HISTORY" parameter.
- The operator selected and program randomized parameters.
- The interrupt and OP CODE counters if requested via "COUNT" parameter. This parameter is reset by the program and must be respecified for subsequent counter displays.
- The random instruction stream in a readable format.
- The interrupt history of the results of the actual instruction stream execution versus the results of the simulation of the instruction stream.
- The control, general purpose and floating point registers associated by the stream.
- Any data area which is referenced by the stream.
- A trace at end of stream does not include a display of the program history.
- A trace after interrupt comparison does not include operator and randomized parameters, or the interrupt and OP CODE counters.
- A data error causes a double display to be made. One area contains the error while the second area is the reference.

8.2.2.1.11 Architecture Table. For each architecture subset permitted via the optional MIL-STD-1750 functions or features, a separate architecture table is provided. This table is a variable length table which contains all the rules of usage of each OP CODE (and OP CODE extensions) supported by that particular architecture. It also identifies those OP CODES (and OP CODE extensions) which may not be used by an architecture. (Note that the command word from the I/O instruction should be considered as an extension to the OP CODE.)

The architecture table is established by means of a series of macros. Once complete, the table is presented to the test program as a directly accessed table.

8.2.3 Costs

The following resources with their respective costs are required to implement the Two Level MIL-STD-1750 certification capability at SEAFAC. Note that the Non-Recurring Start-Up cost of $933,000 and the Recurring cost of $306,000 sums to a total cost of $1,239,000 which is less than the sum of the ATP approach and Random approach alone ($1,470,800 = 617,000 + 853,800) because some duplication of effort is eliminated.

ITEM		Non-Recurring Start-Up Cost (Man Years)
SOFTWARE		
Development Computer System Utilities	=	N.C.
Support Software		
- Cross Assembler for MIL-STD-1750	=	N.C.
- Linkage Loader	=	N.C.
- Simulator	=	N.C.
Modification of AN/AYK-15A ATP	=	3.1
Modification of MIL-STD-1750 Simulator	=	2.0
Development		
- Bootstrap Program	=	0.3
- Random Instruction Generator	=	3.0
- Control Program on Master	=	1.8
- Supervisor Program on Slave for Random	=	1.4
- Source Tape Generator Program	=	0.3
- I/O Test Programs	=	0.5
HARDWARE		
Development Computer	=	N.C.
Master Computer	=	N.C.
MIL-STD-1553 Serial Channel Interface	=	N.C.
RS-232 Serial Channel Interface	=	N.C.
OTHER		
Test Plan Document	=	0.3

SUBTOTAL		12.7 man years
System Integration Factor (5%)		0.63

		13.33 man years/
TOTAL		$933,000

The Recurring cost for 30 computers is $306,000 = ($10.2K * 30). This figure is derived from the following information:

Recurring Cost Item/Computer	Cost (K Dollars)
Hardware Maintenance	0
Software Sustainence	
- ATP	4.14
- Random	2.7
Personnel	
- Coverage to Initialize, Observe, and Analyze Results (2 people for 1 week)	2.8
- Technician to Supervise Integration of I/O Interface	0.28
Other	
- Test Plan to Vendor with Verification Source	0.28
TOTAL	10.20

8.3 CERTIFICATION SCENARIO

After all phases of the MIL-STD-1750 Certification Procedure have been designed, implemented, documented and debugged, the following series of events will take place concerning a vendor certification:

1. A vendor contacts SEAFAC personnel, citing the need for a certification.

2. SEAFAC makes available to the vendor, what requirements must be met to complete the certification (software, GSE, and I/O), providing source modules of the static verification program used in the first phase of testing in the form of a Certification Test Interface Document (see Appendix E).

3. Time is allocated for the certification process to take place.

4. The vendor arrives on site with the MIL-STD-1750 computer and related Ground Support Equipment, and is given time to shakedown and cable-up his hardware.

5. The first phase of verification testing is conducted under supervision of SEAFAC personnel and vendor representatives. Any discrepancies are so noted, all results are documented.

6. The second phase of verification testing is initiated and left to execute for a predetermined length of time. The length of time is proportional to the speed of the unit under test and the number of test cases desired to be run.

7. After a thorough review of all test results, an official statement of compliance or non-compliance is made by SEAFAC concerning the vendor's MIL-STD-1750 computer.

8.4 IMPACTS TO MIL-STD-1750

The recommended approaches do not required any special features to be added to MIL-STD-1750 because of verification tests.

8.4.1 MIL-STD-1750 ARCHITECTURE CONTROL

A complete, detailed, unambiguous specification of any computer architecture (like MIL-STD-1750) is essential to a certification effort in order to ensure that a compatible verification is possible for all vendor implementations.

The architectural specification (MIL-STD-1750) itself is not immutable (fixed) for all time; rather, corrections, modifications and extensions will need to be made over time. Thus, a change mechanism needs to be provided for MIL-STD-1750. Control of potential changes is critical by this change mechanism and must be predicated upon a ground rule that upward compatibility is required. That is, a program written for a MIL-STD-1750 implementation will run and give the same time independent and implementation independent results on an implementation based on MIL-STD-1750A. (Of course, this assumes similar resources for both computers such as 32K storage on both machines.)

The upward compatibility enables the verification program to migrate successfully from an implementation based on MIL-STD-1750 to an implementation based on MIL-STD-1750A. (Of course, the verification program would need to be extended; but deletions or changes would not be required.)

Thus, it is required that different release levels of the verification program would need to be maintained with at least one release level associated with each MIL-STD-1750 release level.

8.4.2 SUBSETS OF MIL-STD-1750

The issue of defining architectural subsets has an important design impact on the certification process's verification approach.

Subset specification guidelines or rules must be provided by the MIL-STD-1750 specification and certification testing must conform to these rules. Since the existence of subsets will require special treatment in the verification approach (as well as either multiple assemblers or a more complex assembler) with subsetting capability, subsetting should be examined carefully. The following is a candidate process for providing member specification:

RULES		OPTION
Must Choose One		Fixed Point
	[ ]	16-Bit Arithmetic
	[ ]	32-Bit Arithmetic
	[ ]	16-Bit and 32-Bit Arithmetic
May Choose One		Floating Point Arithmetic
	[ ]	Short (24/8)
	[ ]	Short and Long
May Choose (options)	[ ]	Expanded Addressing
	[ ]	MIL-STD-1750 I/O Protocol - MIL-STD-1553 Subset
	[ ]	Protection - Memory - Supervisor/Problem States
	[ ]	Start-Up ROM
	[ ]	DMA

Thus, many subsets of the architecture would be permitted following this set of rules. The simplest member would be characterized as a 16-bit fixed-point processor; the other extreme would comprise the entire architecture.

Therefore, multiple architecture subsets require special design considerations for the verification program in order that it <u>properly</u> tests all possible machine implementations.

Parenthetically, the same design considerations is required for support software. For example, the compiler may need to flag floating point equations as illegal for the simplest member. The assembler may likewise need to flag illegal instructions for this simplest member.

8.5 IMPACTS TO SEAFAC

The impact to SEAFAC due to implementing the MIL-STD-1750 certification capability as a two level approach focus on staffing and hardware.

8.5.1 Staffing

Staffing requirements necessary to implement (totally within SEAFAC without any subcontracted work) the recommended verification approaches consist of the following:

Engineer to implement the I/O interfaces

Lead Programme with an architectural background

Support programmers (5)

The staffing required to sustain the recommendation follows:

Engineer/technician to supervise the I/O integration

Programmer/technician to initialize, invoke, and observe the verification program execution

Certification Coordinator

Programmer to maintain the verification programs.

8.5.2 Physical Resources

The physical resources necessary to implement and sustain the recommended verification approaches consists of the following:

Master Computer with Associated Peripherals

Auxiliary Storage (disks)

Magnetic Tape Drives

Printer

Timesharing Operating System

MIL-STD-1553 Interface from the Master Computer

RS-232 Interface from the Master Computer

MIL-STD-1750 Support Software

- Cross Assembler
- Linkage Editor
- Simulator
- Library
- Utilities

Power/Lab Space

9.0 EPILOGUE

IBM is fully confident in the results of this study as documented in this report; however, it is important to recognize that these results are based upon the set of ground rules (developed with Air Force guidance) which are described in Section 2. Different results could be obtained if these ground rules were changed or modified; therefore, it is important to take these results in the context of the ground rules. However, enough information has been provided in this document to permit re-evaluation of the recommendation if the ground rules change.

9.1 OBSERVATIONS/COMMENTS

During the study phase of this contract, many interesting questions were discussed by the IBM team members. This section covers those questions with our responses.

"Since either the MIL-STD-1750 specification will mature over time or else the verification program will mature over time, what happens to a previously certified computer?"

> Retesting needs to be considered. A reasonable scenario for retesting would be to recall all certified computers for retest whenever the verification test is updated. Any discrepancies found would be recorded and published so that users would be aware of the limitations of these existing computers vis a vis the latest level of standard or verification method.

"What advantages or disadvantages arise from supplying vendors with the verification programs?"

> The advantage to the vendor is that each implementation could be pretested prior to the certification process at SEAFAC. This could make the certification process less painful to the vendor. The advantage to the Air Force is that more mature hardware implementations would be submitted for certification testing. An expressed concern is that a vendor would "hand tune" its implementation to only consider the tests in the verification program. If the verification program is of high quality, this should be an advantage, not a concern. Furthermore, IBM believes the two step certification testing process with the Random approach as the second step eliminates this concern.

"For the Manual Test Configuration, isn't it improper for the load tapes to be generated by the vendor?"

In the Manual Test Configuration, the study assumed that the vendor would generate the load tapes for use during the certification process. It was assumed that keeping the load tape for future reference would provide adequate control and protection for the Air Force. It has been suggested that this is an improper approach since merely having the means to establish the validity of the test after having conferred certification is not sufficient in light of the cost, schedule, and competitive damages that could occur. This is acknowledged and is further rationale for not recommending the Manual Test Configuration. Of course, other means of providing the desired level of control can be envisioned such as, requiring a standard load capability for all vendors. However, these alternative methods all have added costs associated with them and further support the recommendations of the study.

"Should a minimum main storage size be required for the certification process?"

Yes, a 32K minimum storage size is required. However, some systems might specify less than 32K storage. For these systems, it would be permitted to let the hardware engineers satisfy the 32K storage requirement by providing a memory expansion connector (which may be inside the box) to provide an external memory extension up to 32K. This expansion could be of commercial quality.

"Should a standard I/O channel like MIL-STD-1553 or RS-232 be required for certification?"

A standard MIL-STD-1553 interface is required. For systems without MIL-STD-1553, the hardware engineers again could provide this feature by providing an I/O connector (which may be inside the box) which connects to an external box providing the MIL-STD-1553 function; or the hardware engineer could provide an RS-232 connector in the box. The external I/O box could be of commercial quality.

"Fcw much error reporting should be provided to the vendor when a trcblem is encountered during certificaticn testirg?"

> The state of the machine at the start of a test case, that specific test case data, the results of that test case after execution upon the machine under test, and the expected results (either predetermined or simulated) should be prcvided for each test failed during the certification process. However, it is not considered the province of the Air Force to prcvide diagnostic information.

"Shouldn't future research be considered for either the Analytical Apprroach or the Computer Hardware Cescriptior Language (ISPS) apprroach?"

> Further consideration should be given to describing the MIL-STD-1750 architecture in these approaches in crder to mature the architecture specificaticn thrcugh a thorcugh simulation effort and to increase the confidence in this architecture.

"Are any additional functions required by the Air Force to further the standardization effort?"

> Perhaps the Air Force could establish a library of operational software functions like SINE, CCSINE, ARCTANGENT, NAVIGATION, KALMAN_FILTER, etc. Perhaps the Air Force cculd provide funds for membership participation in the MIL-STC-1750 Users Group. Perhaps the Air Force could put an organizaticr in place to control, distribute, and maintain Support Software like High Order Languages, Assembler, Simulatcr, Linkage Editor, debugging tccls, etc.

"How does one determine when to stop the Randcm verification approach?"

> A statistical analysis of this problem has been non-productive; engineering judgement has to be used to determine when to stop the Random program.

"Doesn't connecting to the VAX 11/780 strain that resource by being dedicated to the certification process?"

> The VAX 11/780 provides a multi-programmable capability; therefore, the verification program on the VAX 11/780 could run concurrent with other applications.

"When using a Random verification program, how does a test sequence get repeated for a computer which is resubmitted for certification after previously failing the Random tests?"

> The Random verification program should request the base seed at its start. This permits the Generator Program the same pseudo-random sequence of instructions based upon the initialization seed used when the computer first failed.

"How many test cases are required per an average instruction to test the architecture without hardware dependencies being known?"

> Architectural verification programs average 25 tests per instruction in order to thoroughly test the architecture without <u>a priori</u> knowledge of implementation details.

"What testing philosophy should be used regarding exceptional (error) conditions?" (e.g., Store Multiple (STM) crossing page protection boundaries, an unnormalized floating point operand.)

> Not only should testing for normal conditions occur, but even more important is the testing of the exceptional (error) conditions and the status of the machine upon completion of the error event. e.g., a Store Multiple crossing page boundaries from an unprotected page into a protected page should terminate the same way on any machine being tested; that is, STM should terminate without storing into the protected page and this should be so tested.

"What testing philosophy should be used regarding "Required", "Optional", "Reserved" and "Spare" features in MIL-STD-1750?"

> "Required", "Optional", and "Reserved" define architectural features which must be tested; "Spare" is not defined and cannot be tested.

"What testing philosophy should be applied to implied conditions in the MIL-STD-1750 architecture?"

An example of an implied condition is storing only the IM, SW, and IC into main storage when an interrupt occurs. In this case, the adjacent locations should also be tested to guarantee they did not change (i.e., defining a four word PSW is illegal, therefore, implied conditions should also be tested.

THIS PAGE INTENTICNAIIY IEFT BLANK

10.0 APPENDIX A - SEAFAC FACILITIES

The System Engineering Avionics Facility (SEAFAC) is an avionic hot-bench and test facility within Avionics Engineering Directorate (ENA) Aeronautical Systems Division (ASD) of the Air Force at WPAFB. Originally organized to analyze, design, simulate, and evaluate avionic equipment in a total system context, the facility has been tasked to play a key role in the standardization effort of the Air Force. Through SEAFAC, the capability to aid users in the application of standards and to verify compliance with the standards is being developed on an evolving basis to meet the requirements of the Air Force.

With respect to standards verification, SEAFAC has been the principal agency for the MIL-STD-1553 bus effort. Three generations of bus testers have been designed to provide both in-house and on-site certification and system engineering of MIL-STD-1553 equipment. A number of projects have been addressed in support of Program Office activities which has expanded the role of SEAFAC in providing benefits and capabilities to ASD.

The resources available at SEAFAC consists of:

1. Physical Plant - Facilities for administrative support and capabilities to accommodate hot-bench system configurations, hardware design and fabrication, and testing and evaluation are available.

2. Electronic Hardware - A variety of hardware resources exist, including:

   a. PDP-11/55 computer system with links to a MIL-STD-1553 data bus. Mass memory is provided by two RK-05 disc drives (each containing 2.5 megabytes), two RK-06 disc drives (each containing 13.9 megabytes), a dual floppy disc drive, and a 112K word memory.

   b. PDP-11/34M with 32 kilowords of main memory and with custom designed bus interface unit for bus controller or remote terminal operation on a MIL-STD-1553 bus.

   c. VAX 11/780 computer system with 1 million bytes of memory, two RM03 disc drives (each containing 64 megabytes), 2 switchable shared RK-06 disc drives.

   d. Microcomputer development system (e.g., IMP16, SILENT 700, 8080).

   e. Remote terminals for access to ASD computer center containing a CYBER 175 and System 360/370 emulator.

f. Minicomputer (CP-16A and AYK-8).

g. Test equipment including meters, scopes, generators, logic analyzers, custom designed multiplex data bus testers, and MIL-STD-1553 verification equipment.

3. Software - Support software for the PDP-11/55 consists of the RSX-11M operating system, Macro-11 assembler, and FORTRAN IV - PLUS compiler.

4. Personnel - The SEAFAC personnel contingent is comprised of hardware engineers, software programmers, and technicians with experience in configuring and implementing hot-bench avionic system setups. This branch of the Aeronautical Systems Division is broken into three working groups; Hardware, Software and Multiplex. The staffing level for each of these groups is as follows:

Hardware	
Engineers	6
Technicians	3
Software	
Programmers	3
Technicians	2
Multiplex	
Engineers	4
Technicians	1

20.0 APPENDIX B - ESTIMATION OF THE TOTAL NUMBER OF ARCHITECTURAL DISCREPANCIES

This section describes in detail the method for projecting the total number of architectural discrepancies. As discussed previously, Engineering Changes (ECs) are a measure of the total number of architectural discrepancies which have been found to date for a given machine design. It is desirable to know the total number of architectural discrepancies which remain in a machine at the time of sell-off, i.e., those already found plus those which remain undiscovered.

It is expected that the rate of discovery of architectural discrepancies will be an exponentially decreasing function of time. Initial use of the machine will be for the development and validation of operational software. This new software will exercise the machine in new and previously untested ways. It is expected that most discrepancies will be discovered early in this usage. As the application programs become more developed, fewer and fewer new exercises of the machine are performed, and the rate of error discovery declines. Finally, after the applications programs are completely developed and in use for some time, the rate of discrepancy discovery approaches zero. This may be plotted as shown in Figure 20-1, where

f(t) is the number of ECs discovered as a function of time.

The area under this decreasing exponential curve (i.e., the integral) is the total number of architectural discrepancies which remain in the machine after sell-off, which is the desired quantity.

The curve is of the form

$$f(t) = ae^{-bt} . \qquad (1)$$

To calculate the area under the curve, let C equal the total number of architectural discrepancies. Then

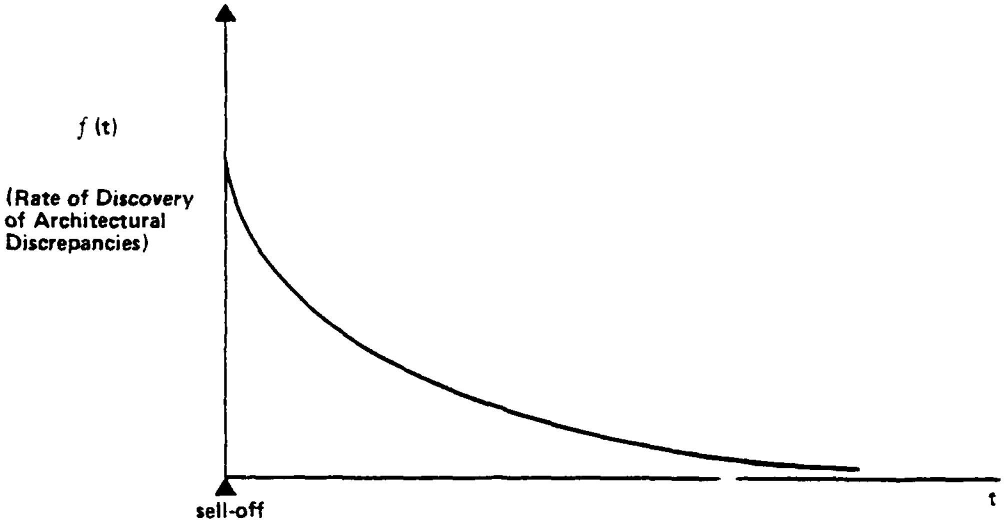


Figure 20-1. Expected Number of Architectural Discrepancies Over Time

$$C = \int_0^\infty f(t)\ dt = \int_0^\infty a e^{-bt} \qquad (2)$$

$$= -\frac{a}{b} e^{-bt} \Big]_0^\infty \qquad (3)$$

$$= \frac{a}{b} \qquad (4)$$

20.1.1 The Decreasing Exponential Method

In order to compute C, it is necessary to fit a curve (with the correct values of a and b) to the data, $\hat{f}(t)$, which might appear as is shown in Figure 20-2.

This may be accomplished by taking the natural log transform of the data, thereby making it linear and then performing linear regression analysis to obtain the best fitting line. This results in a semi-logarithmic curve fit. (This will be called the decreasing

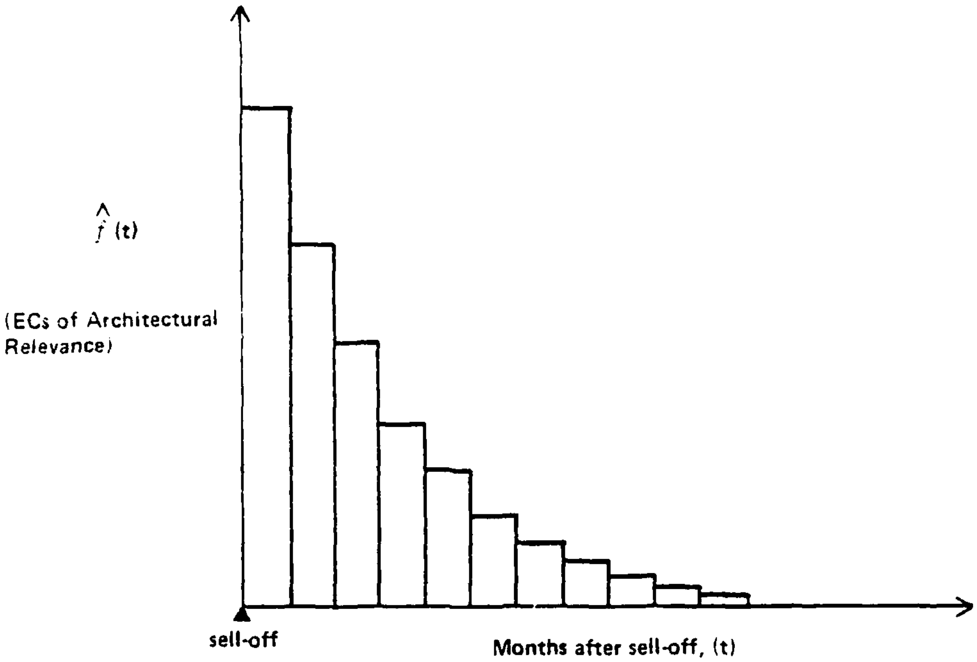


Figure 20-2. Expected Data: ECs after Sell-off versus Time

exponential method.) The transform is

$$\ln(\hat{f}(t)) = \ln(ae^{-bt}) = \ln(a)-bt \qquad (5)$$

The right-hand side of this equation is in the form of a straight line, for which the general expression is y = mx + b, where m is the slope and b is the y-intercept. Linear regression analysis yields the slope, in this case b, and the intercept, in this case ln(a), which are the desired variables. To obtain the value of C, the value of b and the anti-log of ln(a) are taken and applied to Equation (4).

The slope and intercept are calculated as follows:

$$\text{slope} = b = \frac{\left[\dfrac{\sum_{i=1}^{N} t_i * \hat{f}(t_i)}{N} \right] - t * \hat{f}(t)}{\sigma_t^2} \qquad (6)$$

$$\text{intercept} = \ln(a) = \overline{\hat{f}(t_i)} - b\,\overline{t_i} \qquad (7)$$

where t_i are the months after sell-off and $\hat{f}(t_i)$ are the corresponding number of ECs of architectural relevance for those months. N is the total number of data points. The average value of $\hat{f}(t_i)$, represented by $\overline{\hat{f}}(t_i)$, is

$$\overline{\hat{f}(t_i)} = \frac{\sum_{i=1}^{N} \hat{f}(t_i)}{N} \qquad (8)$$

The average value of t_i, represented by $\overline{t_i}$, is

$$\overline{t_i} = \frac{\sum_{i=1}^{N} t_i}{N} \;. \qquad (9)$$

The variance of the t values, σt^2, is

$$\sigma_t^2 = \frac{\sum_{i=1}^{N} t_i^2}{N} - \overline{t}^2 \qquad (10)$$

As an application of this method, the data from a System/370 model may be evaluated. (It is not intended that the results be evaluated in this section. This evaluation appears in Section 7. The data are brought here for the purpose of illustration.) The data is shown in Table 20-1 and is plotted in Figure 20-3. The natural log transform of the data is taken, using Equation (5), and the best fit line is found using Equations (6),(7),(8),(9), and (10). These are plotted in Figure 20-4.

Table 20-1. ECs of Architectural Relevance From A System/370 Model

Month (t)	ECs ($\hat{f}(t)$)
1	4
2	4
3	3
4	3
5	5
6	2
7	5
8	3
9	1
10	4
11	1
12	0
13	2
14	4
15	1
16	2
	Total 44

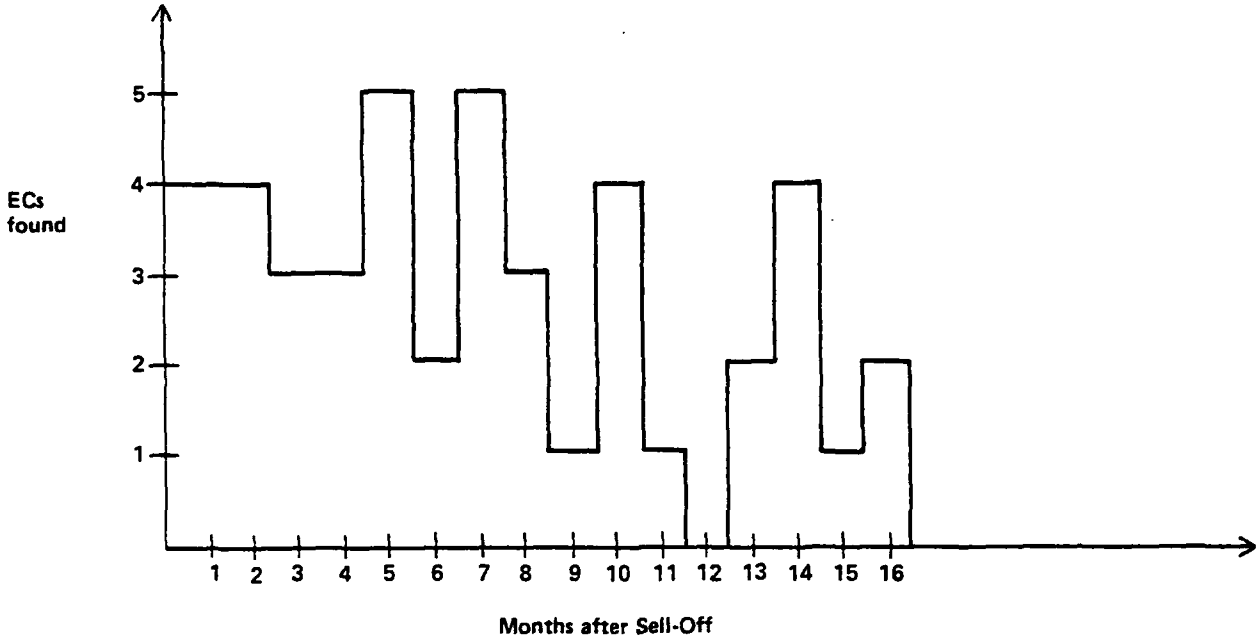


Figure 20-3. ECs of Architectural Relevance for System/370 Model versus Time

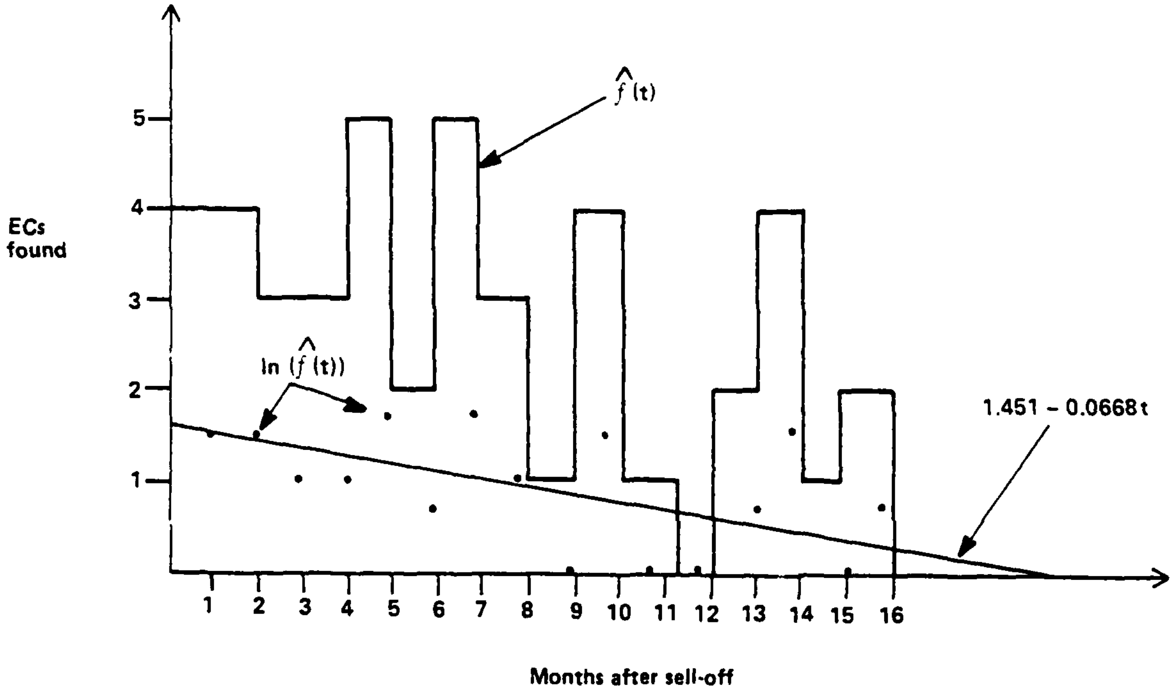


Figure 20-4. ECs of Architectural Relevance for System/370 Model versus Time and Transformed Data with Best Fit Line

Note that no ECs were found in month twelve. This presents a problem since ln(0) equals minus infinity. For this analysis, the twelfth month was assumed to have one EC. This problem will be discussed in more detail later.

The straight line found is

$$\ln(\hat{f}(t)) = 1.451 - 0.0668\ t \qquad (11)$$

This yields the desired curve, $\hat{\hat{f}}(t)$, by taking the anti-log of Equation (11).

$$\hat{\hat{f}}(t) = e^{\ln(\hat{f}(t))} = e^{1.451 - 0.0668t} = 4.267\ e^{-0.0668t} \qquad (12)$$

This is plotted in Figure 20-5.

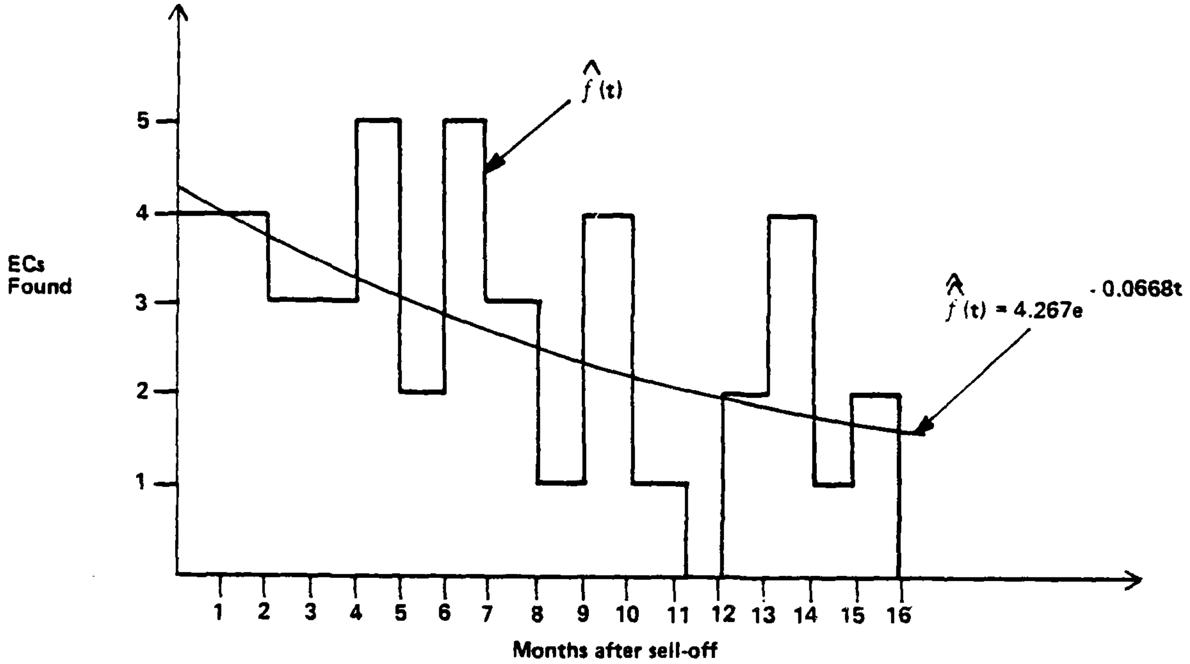


Figure 20-5. ECs of Architectural Relevance for System/370 Model versus Time and Best Fit Curve

A total of 44 ECs were found by the end of the 16th month as shown in

Table 20-1. The total projected number cf architectural discrepancies is found as follows:

$$C = \frac{a}{b} = \frac{4.267}{0.0668} = 63.88, \text{ or approximately } 64 \qquad (13)$$

In other words, 20 additional discrepancies are projected, but remain undiscovered.

As menticned previously, a difficulty with this <u>decreasing exponential</u> method is its inability to handle data points of value zero, since the natural log of zero is minus infinity.

20.1.2 <u>The Cumulative Data Approach</u>

An alternative approach is to use the cumulative function, which has only non-zero values.

The cumulative function, denoted by g(t), is the integral of the original data function, f(t).

$$g(t) = \int f(t)\,dt = \int a\,e^{-bt}\,dt$$

$$= a\int e^{-bt}\,dt$$

$$= -\frac{a}{b}\,e^{-bt} + C_o, \qquad \text{or using Equation (4)}$$

$$= C_o - C\,e^{-bt} \qquad (14)$$

It is known that at t = 0, no ECs have been found. Therefore g(0) = 0 at t = 0. Using Equaticn (14),

$$g(0) = C_0 - C e^{-b \times 0}$$

$$0 = C_0 - C e^{0} \qquad (15)$$

$$C_0 = C$$

The resulting form of the cumulative function is:

$$g(t) = C - C e^{-bt} . \qquad (16)$$

This is plotted in Figure 20-6.

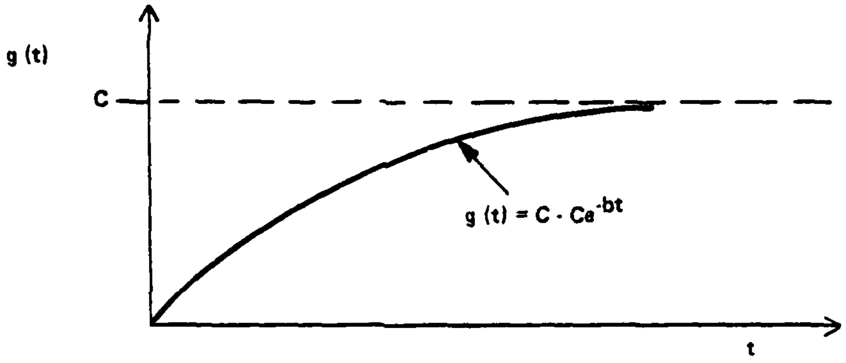


Figure 20-6. Cumulative ECs Over Time

The asymptote, C, is the total projected number of architectural discrepancies.

The goal once again is to fit the best curve to the actual data, i.e, determine the best values of C and b. An example of what the cumulative data, $\hat{g}(t)$, might look like is shown in Figure 20-7.

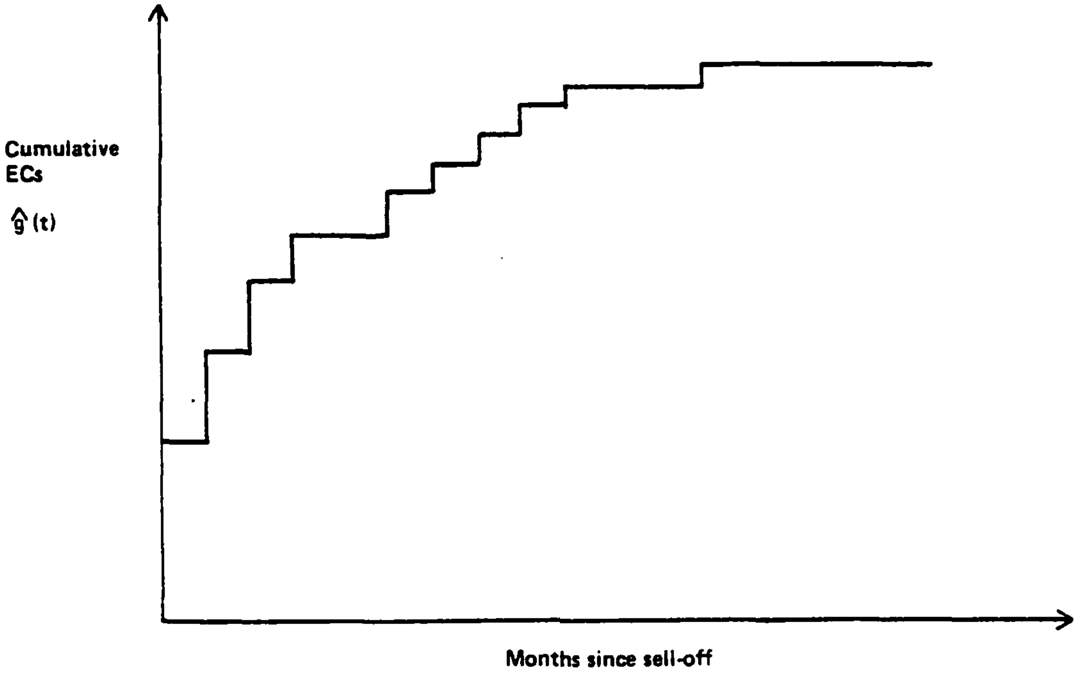


Figure 20-7. Cumulative ECs Over Time, Data

It would be desirable to take the log transform of the data (to make it linear), fit a straight line by linear regression, and then take the anti-log to obtain the curve, as was done previously with f(t). Unfortunately, the natural log of $C-Ce^{-bt}$ does not produce a straight line. Note that the derivative (slope) is

$$\frac{d}{dt} \ln (C - C e^{-bt}) = \frac{1}{C - C e^{-bt}} \quad C b e^{-bt}$$

$$= \frac{b e^{-bt}}{1 - e^{-bt}}$$

Since the slope of $\ln (C - C e^{-bt})$ is not a straight line, but is a function of t , linear regression analysis cannot be used. Therefore, the following approach to fitting the best curve (choosing the best values of C and b) is used.

First, an arbitrary value is chosen for C and the data, g(t), is subtracted from it. This yields

$$h(t) = C - g(t) = C e^{-bt} \tag{18}$$

which is plotted in Figure 20-8.

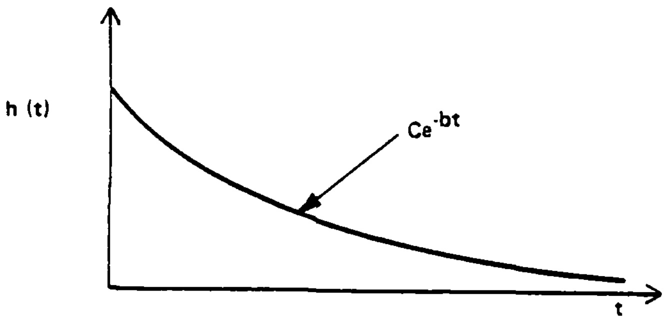


Figure 20-8. C Minus the Cumulative EC Function

Next, the natural log is taken.

$$\ln h(t) = \ln C - bt \tag{19}$$

This is in the form of a straight line, which is plotted in Figure 20-9.

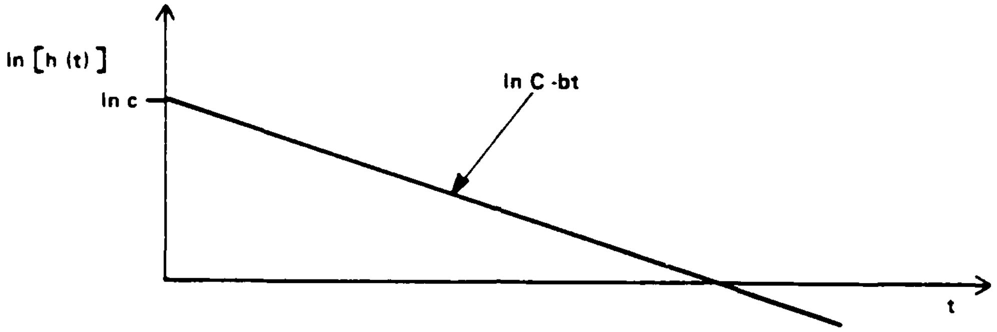


Figure 20-9. Log Transform of Cumulative Data

Since this is in the form of a straight line, linear regression analysis may be applied to determine the line that best fits the data. Equations (6), (7), (8), (9), and (10) are applied, except that $f(\hat{t}_i)$ is replaced with $[\ln \hat{h}(t_i)]$.

The slope, b, has thereby been determined and, since C was chosen, the equation for the curve is complete.

The difficulty remains that C was chosen arbitrarily. The solution is to choose different values of C and then determine the goodness of fit between the resulting curve, $\hat{\hat{g}}(t) = C - C e^{-bt}$, and the data, $\hat{g}(t)$. The better the fit, the better the estimate of C. This process is then iterated until the best estimate of C is obtained.

Two different measures of the degree of fit can be used: the correlation coefficient (r) of the line calculated by regression analysis, and the squared error of the calculated cumulative curve, $\hat{\hat{g}}(t)$, and the cumulative data, $\hat{g}(t)$.

The correlation coefficient is a measure of the degree of association between the random variables $(\ln[\hat{h}(t_i)], t_i)$ .... $(\ln[\hat{h}(t_n)], t_n)$. It is denoted by r and is calculated by using the following expression:

$$r = \frac{\dfrac{\sum_{i=1}^{N} \ln\left[\hat{h}(t_i)\right] * t_i}{N} - \overline{\ln\left[\hat{h}(t)\right]} * \bar{t}}{\sigma_{\ln\left[\hat{h}(t)\right]} * \sigma_{t_i}} \qquad (20)$$

Note that this applies to the log transform of the data, i.e., the straight line. The higher the r-value, the better the fit.

The squared error is a measure of the fit of the resulting curve with the data. It is calculated by the following formula

$$\text{Squared Error} = \sum_{i=1}^{N} \left[\hat{\hat{g}}(t_i) - \hat{g}(t_i)\right]^2 . \qquad (21)$$

The curve shown in Figure 20-10 helps in visualizing the squared error concept.

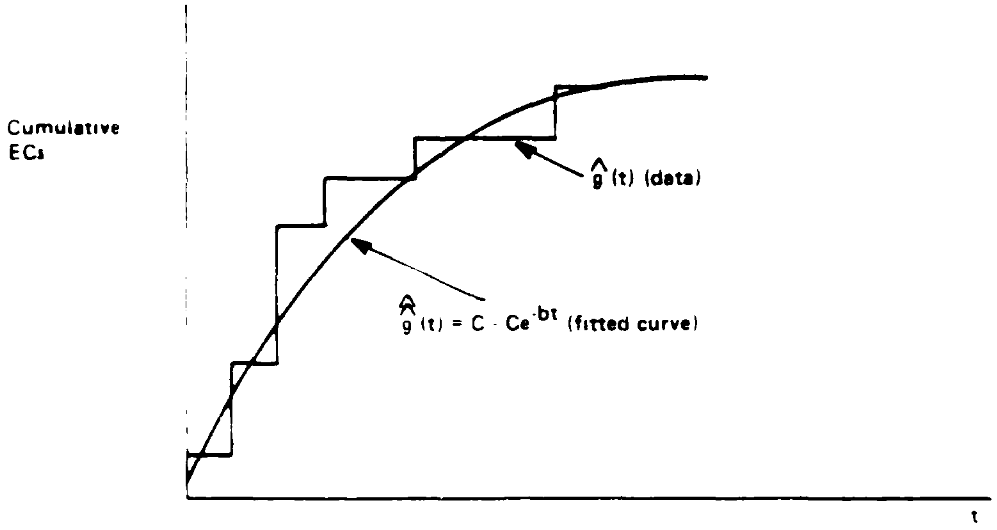


Figure B-10. Squared Error Representation

For each value of t_i, the difference between the fitted curve and the data is squared. These terms are then summed. The better the fit of the curve to the data, the lower the total squared error.

To summarize, the value of C, which is the total projected number of architectural discrepancies which remain in the machine after sell-off, may be estimated by use of an iterative procedure. This procedure involves selecting various arbitrary values of C until the best estimate of C is found, based on a maximum r value or a minimum squared error value.

B.4 NORMALIZATION OF DATA BY MACHINE SIZE

As was discussed previously, the projected total number of architectural discrepancies must be normalized by machine size in order that a fair estimate of quality be obtained for each of the certification methods. That is, the purpose is to compute the expected number of architectural discrepancies of a verification method for a machine of the nominal size expected for MIL-STD-1750.

The simplest measure of machine size is the number of gates used in the machine. Unfortunately, this ignores the microcode and main storage of the machine, either of which (if working incorrectly) would result in an architectural discrepancy. It would be possible to simply sum the sizes of the logic, microcode and main storage. However, it is expected that equal weighting of a gate of logic, a bit of microcode, and a bit main storage would exaggerate the importance of main storage. Therefore, each of the three components is normalized separately. To accomplish this, each of the ECs of architectural relevance uncovered in this study is examined to determine whether its cause is logic-related, microcode-related, or main storage-related.

The projected number of architectural discrepancies remaining in the machine after the application of Verification Method A in the nominal size MIL-STD-1750 machine (denoted by DISCREP(1750)) is the sum of three components:

$$\begin{aligned}\text{DISCREP(1750)} &= \text{Logic DISCREP(1750)} + \text{Microcode DISCREP(1750)} \\ &\quad + \text{Main Storage DISCREP(1750)} \qquad \text{(B1)}\end{aligned}$$

Assume that the architectural discrepancy data were gathered from Machine B, which is different in size than the nominal MIL-STD-1750 machine. The first component in the above equation, the number of logic-related architecture discrepancies expected for a nominal sized MIL-STD-1750 machine after the use of Verification Method A, becomes

$$\text{Logic DISCREP(1750)} = \text{Logic DISCREP(B)} \times \begin{array}{l}\text{machine size} \\ \text{normalization} \\ \text{factor.}\end{array} \qquad \text{(B2)}$$

Where Logic DISCREP(B) refers to the projected number of architectural discrepancies associated with logic found in machine B.

The machine size normalization factor in Equation (B2) is simply

$$\begin{array}{l}\text{machine size} \\ \text{normalization} \\ \text{factor}\end{array} = \frac{\text{size of nominal 1750 machine (gates)}}{\text{size of machine B (gates)}}$$

The microcode and the main storage components are treated in a similar fashion.

As an example of the use of this method, the nominal MIL-STD-1750 machine was expected to contain 30,000 gates, 200,000 microcode bits, and 1,000,000 main storage bits. Next, suppose that Verification Method A was used on a large machine (B) and machine (B) contained

100,000 gates, 400,000 microcode bits and 10,000,000 main storage bits. Also suppose that 50 architectural discrepancies were projected. Of these, 30 were logic-related, 15 were microcode-related, and 5 were main storage-related. The projected number of architectural discrepancies for Verification Method A for a nominal MIL-STD-1750 machine becomes:

$$\begin{aligned}
\text{DISCREP(1750)} &= 30 \text{ logic DISCREP(B)} \times \frac{\text{30,000 gates (1750)}}{\text{100,000 gates (B)}} \\
&+ 15 \text{ microcode DISCREP(B)} \times \frac{\text{200,000 bits (1750)}}{\text{400,000 bits (B)}} \\
&+ 5 \text{ main storage DISCREP(B)} \times \frac{\text{1,000,000 bits (1750)}}{\text{10,000,000 bits (B)}} \\
&= 30 \times 0.3 + 15 \times 0.5 + 5 \times 0.1 \\
&= 17
\end{aligned}$$

Verification Method A would be expected to miss 17 (not 50) architectural discrepancies when applied to a machine of the nominal size expected to be used in MIL-STD-1750 applications.

Normalization has prevented an over-estimation of the total number of architectural discrepancies for Verification Method A simply because it was used on a large machine (B).

30.0 APPENDIX C - PROGRAMS FOR ESTIMATING TOTAL NUMBER OF ARCHITECTURAL DISCREPANCIES

```
  ∇ESTIMATE[⎕]∇
  ∇ ESTIMATE
 ⍝
 ⍝ ESTIMATE C (TOTAL NUMBER OF EC'S) BY SUCCESSIVE APPROXIMATION
 ⍝  (0) INITIALIZATION --- CHI ← 200  CLO ← TOTAL NUMBER OF KNOWN EC'S

 ⍝  (1) ESTIMATE C: MIDPOINT OF CHI AND CLO
 ⍝  (2) FIT CURVE TO DATA (USING ESTIMATE OF C)
 ⍝  (3) FIT CURVES USING C+1 AND C-1
 ⍝  (4) IF SQUARED ERROR OF CURVE FOR C ≤ SQUARED ERROR OF CURVES FOR
 ⍝      C+1 AND C-1 THEN C IS BEST ESTIMATE: STOP
 ⍝  (5) IF SQUARED ERROR OF CURVE FOR C-1 < SQUARED ERROR OF
 ⍝      CURVES FOR C AND C+1 THEN CHI ← C+1; GO TO (1)
 ⍝  (6) IF SQUARED ERROR OF CURVE FOR C+1 < SQUARED ERROR OF
 ⍝      CURVES FOR C AND C-1 THEN CLO ← C-1; GO TO (1)
 ⍝
  ⎕←' '
  ⎕←' '
  ⎕←' '
  ⎕←'                             A-7 ENGINEERING CHANGES'
  ⎕←'        '
  X←⍳(⍴Y)
  ⎕←'MONTHS SINCE FIRST SHIP: ', 3 0 ⍕X[⍳18]
  ⎕←'CUMULATIVE EC''S:          ', 3 0 ⍕Y[⍳18]
  ⎕←'    '
  ⎕←'MONTHS SINCE FIRST SHIP: ', 3 0 ⍕X[18+⍳17]
  ⎕←'CUMULATIVE EC''S:          ', 3 0 ⍕Y[18+⍳17]
  ⎕←' '
  CHI←200
  CLO←Y[(⍴Y)-1]
  ⎕←'ESTIMATE OF C       SQUARED ERROR    '
LOOP1:CMID←⌊(CHI+CLO)÷2
  CM1←CMID-1
  CM2←CMID+1
```

```
[32]  C←CMID
[33]  X FIT Y
[34]  SQMID←SQERR
[35]  ⎕← 7 0 25 5 ⍕CMID,SQMID
[36]  C←CM1
[37]  →(C>Y[(⍴Y)-1])/L0
[38]  C←C+0.1
[39] L0:X FIT Y
[40]  SQM1←SQERR
[41]  C←CM2
[42]  X FIT Y
[43]  SQM2←SQERR
[44]  →((SQMID≤SQM1)∧(SQMID≤SQM2))/FINISH
[45]  →(SQMID>SQM1)/L1
[46]  CLO←CM1
[47]  →LOOP1
[48] L1:CHI←CM2
[49]  →LOOP1
[50] FINISH:⎕←' '
[51]  ⎕←'TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES'
[52]  ⎕←'(SQUARED ERROR TECHNIQUE) = ',⍕CMID
[53]  ⎕←' '
[54]  ⎕←' '
[55]  ⎕←' '
      ∇
```

```
        ∇FIT[⎕]∇
        ∇ X FIT Y
[1]     ⍝
[2]     ⍝ FIT EXPONENTIAL CURVE TO DATA (C - Y):
[3]     ⍝ (0) LINEARIZE DATA: LN (C - Y)
[4]     ⍝ (1) COMPUTE LINEAR LEAST SQUARED ESTIMATE FOR LINEARIZED DATA
[5]     ⍝ (2) SLOPE OF RESULTING LINE IS EXPONENT FOR EXPONENTIAL
[6]     ⍝ (3) COMPUTE SQUARED ERROR BETWEEN DATA AND EXPONENTIAL CURVE
[7]     ⍝
[8]     Z←⍟(C-Y)
[9]     MZ←MEAN Z
[10]    MX←MEAN X
[11]    VARZ←VAR Z
[12]    VARX←VAR X
[13]    STDVZ←VARZ*0.5
[14]    STDVX←VARX*0.5
[15]    CROSSSUM←+/(Z×X)
[16]    ⍝
[17]    ⍝ COMPUTE R, THE CORRELATION COEFFICIENT
[18]    ⍝
[19]    R←((CROSSSUM÷(⍴Z))-(MZ×MX))÷(STDVZ×STDVX)
[20]    SLOPEZ←(R×STDVZ)÷STDVX
[21]    ⍝
[22]    ⍝ F(C,SLOPEZ) IS THE EQUATION OF THE CUMULATIVE EC CURVE
[23]    ⍝
[24]    SQERR←+/((Y-(C F SLOPEZ))*2)
        ∇

        ∇F[⎕]∇
        ∇ Z←C F A
[1]     Z←C×(1-*(A×X))
[2]
        ∇
```

```
      ∇VAR[⎕]∇
      ∇ Z←VAR X
[1]   ⍝
[2]   ⍝ COMPUTE THE VARIANCE OF X
[3]   ⍝
[4]    Z←((+/(X*2))÷(⍴X))-((MEAN X)*2)
      ∇
```

```
      ∇MEAN[⎕]∇
      ∇ Z←MEAN X
[1]   ⍝
[2]   ⍝ COMPUTE THE MEAN OF X
[3]   ⍝
[4]    Z←(+/X)÷⍴X
      ∇
```

40.0 APPENDIX D - ESTIMATES OF TOTAL ARCHITECTURAL DISCREPANCIES

B-52D SPN/GEANS ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:	0	1	2	3	4	5	6
CUMULATIVE ECs:	0	15	17	19	20	25	26

ESTIMATE OF C	SQUARED ERROR
113	320.35137
70	272.04194
48	207.97311
37	140.82952
32	91.88000
29	54.96710
28	42.76363
27	34.35369

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(SQUARED ERROR TECHNIQUE) = 27

B-52D SPN/GEANS ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:	0	1	2	3	4	5	6
CUMULATIVE ECs:	0	15	17	19	20	25	26

ESTIMATE OF C	CORRELATION COEFFICIENT
113	-.91590
70	-.92712
48	-.94218
37	-.95714
32	-.96556
29	-.96672
30	-.96736

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(CORRELATION COEFFICIENT TECHNIQUE) = 30

```
                              A-7 ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:    0   1   2   3   4   5   6   7   8   9  10  11  12
CUMULATIVE ECs:             0   0   2   2   2   6   9   9  11  12  13  13  14

MONTHS SINCE FIRST SHIP:   13  14  15  16  17  18  19  20  21  22  23  24  25
CUMULATIVE ECs:            14  15  15  16  17  17  20  20  20  22  22  22  22

MONTHS SINCE FIRST SHIP:   26  27  28  29  30  31  32  33  34
CUMULATIVE ECs:            22  22  22  22  22  22  24  24  24

ESTIMATE OF C          SQUARED ERROR
    112                 431.75694
     68                 323.24758
     46                 196.48119
     35                  91.16330
     30                  46.95553
     27                  59.01848
     28                  46.65670
     29                  44.04095

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(SQUARED ERROR TECHNIQUE) = 29

                              A-7 ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:    0   1   2   3   4   5   6   7   8   9  10  11  12
CUMULATIVE ECs:             0   0   2   2   2   6   9   9  11  12  13  13  14

MONTHS SINCE FIRST SHIP:   13  14  15  16  17  18  19  20  21  22  23  24  25
CUMULATIVE ECs:            14  15  15  16  17  17  20  20  20  22  22  22  22

MONTHS SINCE FIRST SHIP:   26  27  28  29  30  31  32  33  34
CUMULATIVE ECs:            22  22  22  22  22  22  24  24  24

ESTIMATE OF C          CORRELATION COEFFICIENT
    112                  -.95859
     68                  -.96407
     46                  -.97126
     35                  -.97852
     30                  -.98295
     27                  -.98381
     28                  -.98411

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(SQUARED ERROR TECHNIQUE) = 29
```

```
                     S/370 MODEL ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:   0  1  2  3  4  5  6  7  8  9 10 11 12
CUMULATIVE ECs:            0  4  8 11 14 19 21 26 29 30 34 35 35

MONTHS SINCE FIRST SHIP:  13 14 15 16
CUMULATIVE ECs:           37 41 42 44

ESTIMATE OF C         SQUARED ERROR
    122                 97.82478
     83                 33.64729
     64                 19.76986
     73                 19.42355
     68                 16.81260

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(SQUARED ERROR TECHNIQUE) = 68

                     S/370 MODEL ENGINEERING CHANGES

MONTHS SINCE FIRST SHIP:   0  1  2  3  4  5  6  7  8  9 10 11 12
CUMULATIVE ECs:            0  4  8 11 14 19 21 26 29 30 34 35 35

MONTHS SINCE FIRST SHIP:  13 14 15 16
CUMULATIVE ECs:           37 41 42 44

ESTIMATE OF C         CORRELATION COEFFICIENT
    122                 -.99267
     83                 -.99538
     64                 -.99637
     73                 -.99615
     68                 -.99639
     66                 -.99641

TOTAL PROJECTED ARCHITECTURAL DISCREPANCIES
(CORRELATION COEFFICIENT TECHNIQUE) = 66
```

THIS PAGE LEFT INTENTIONALLY BLANK

50.0 APPENDIX E - CERTIFICATION INTERFACE DOCUMENT

50.1 DESCRIPTION

The Certification Interface Document serves as a means of communication between SEAFAC and the vendor for the certification process. This document contains detailed information of the following type:

- Certification System Hardware Configuration
- Certification Scenario
  - Schedule of Events
- Physical Resources Available to the Vendor
  - Power
  - Space
  - Cooling
  - Access Times
- Vendor Provided Hardware
  - MIL-STD-1750 Computer to be Tested
  - MIL-STD-1553 I/O Channel
  - Cables
- Vendor Provided Software
  - I/O Subroutine Descriptions
- SEAFAC Provided Software
  - Support Software
  - Certification Program Source (two - AVP and Random)
  - Bootstrap Program Source
- Vendor Certification Personnel Requirements

- SEAFAC Certification Perscnnel
  - Observer
  - Technician
  - Coordinator

60.0 APPENDIX F - BIBLIOGRAPHY

Barbacci, M.R., A User's Guide to the ISPL Compiler, 12 January 1976.

Barbacci, M.R., Instruction Set Processor Specifications (ISPS) the Notation and Its Applications, 17 May 1979, CMU-CS-79-123 Carnegie-Mellon University, Pittsburg, PA.

Barbacci, M.R., ISP Description of Four Military Computer Architectures.

Barbacci, M.R., Dietz, W.B., Szewerenko, L., Specification, Evaluation and Validation of Computer Architectures Using Instruction Set Processor Descriptions, 13 April 1979, CMU-CS-79-118 Carnegie Mellon University, Pittsburg, PA.

Birman, A., Correctness in Design: The S-Machine Experiment, 17 January 1973, RC4193.

Birman, A., Joyner, W.H., MVS - A System for Microprogram Validation, Part 1 - The Skeleton, July 1974, RC-4923.

Blikle, A., Budkowski, S., Certification of Microprograms by an Algebraic Method, 9th Annual Workshop on Microprogramming, New Orleans, LA, 9-14, 27-29 September 1976.

Boswell, F.R., Automated Testing of Digital Assemblies and Subsystems Semiconductor Integrated Circuit Processing and Production Conference (Abstracts), Anaheim, CALIF., 47. 2PP, Industrial and Science Conference Management, 9-11 February 1971.

Bouricius, W.G., Procedure for Testing Microprograms (Revised). September 1974, RC-5017.

Brand, D., Algebraic Simulation Between Parallel Programs, June 1973, RC-7206.

Brand, D., Joyner, W.H.,Jr., Verfication of Protocols using Symbolic Execution, Computer Networks (Netherlands), Vol. 2, No. 4-5, 351-60, 20.

Budkowski, S., Dembinski, P., Firmware Versus Software Verification, Proceedings of the 11th Annual Microprogramming Workshop, Pacific Grove, CA., 119-27, 19-22 Nov. 1978.

Burr, William E., Fuller, Samuel H., Shaman, Paul S., Lamb, David A., Computer Family Architecture Selection Committee Final Report. Volume III, Evaluation of Computer Architecture via Test Programs.

Carter, W.C., Joyner, W.H.,Jr., Brand, D., Ellozy, H., Wolf, J.L., An Improved System to Verify Assembled Programs, FTCS-8, the Eight Annual International Conference on Fault-Tolerant Computing, Toulouse, France, 165-70, 21-23, June 1978.

Carter, W.C., Ellozy, H.A., Joyner, W.H.,Jr., Leeman, G.B.,Jr., Techniques for Microprogram Validation, Integrity in Electronic Flight Control Systems, 9/1-19, 32.

Carter, W.C., Joyner, W.H., Leeman, G.B., Techniques to Simplify Microcode Validation, 1976 International Symposium on Fault-Tolerant Computing, Pittsburgh, PA, 196, 21-23, June 1976.

Carter, W.C., Joyner, W.H., Brand, D., Microprogram Verification Considered Necessary, April 1978, RC-7053.

Carter, W.C., Joyner, W.H., Ellozy, H.A., Leeman, G.B., Techniques for Microprogram Validation, January 1977, RC-6361.

Case, G.P., A Statistical Method for Test Sequence Evaluation, 12th Design Automation Conference, Boston, MASS., 257-8, 23-25 June 1975.

Cleemput, W.M., Conference Chairman Proceedings of the 4th International Symposium on Computer Hardware Description Languages, October 1979, IEEE Catalogue No. 79CH1436-5C.

Dembinski, P., Budkowski, S., An Introduction to the Verification Oriented Microprogramming Language 'Middle', Proceedings of the 11th Annual Microprogramming Workshop, Pacific Grove, CA., 139-43, 19-22 November 1978.

Durgavich, J.J., Granieri, M.N., Woodpine, J.J., ATE System Architecture Alternatives, Autotesting '78, International Automatic Testing Conference, San Diego, CA., 199-209, 28-30 Nov. 1978.

Elspas, B., Green, M.W., Levitt, K.N., Waldinger, R.J., Research in Interactive Program-Proving Techniques, May 1972, Stanford Research Institute, Menlo Park, California.

Evangelisti, C.J., Goertzel, G., Lesser, J.D., Oflk, R., Structured Sequential Machine Comparison, IBM Technical Disclosure Bulletin, Vol. 20, No. 6. 2500-7. 0.

Fohl, M.E., Central Processor Diagnosis by Function, Digest of Papers from the 1972 International Symposium on Fault-Tolerant Computing, Newton, MASS., 62-7, 2, 19-21 June 1972.

Guffin, R.M., Microdiagnostics for the Standard Computer MLP-900 Processor, IEEE Trans. Comput, Vol. 20, No.7. 803-8.

Hedeman, W.R., III, Program to Count Microcode Instructions, IBM Technical Disclosure Bulletin, Vol. 16, No. 1. 290-1. 0.

Hoehne, H., Piloty, R., Design Verification at the Register Transfer Language Level, IEEE Trans. Comput., Vol.C-24, No.9. 861-7. 17.

Inagaki, M., Test and Diagnosis Program Generation Using Microinstruction, Nec. Res. and Devep. (Japan) No. 26. 35-52. 6.

Jacobowitz, H., Automatic Generation of Tests and Diagnostics for Chips, Plug-Ins and Computer Systems, Computer Designer's Conference and Exhibition (abstracts), Anaheim, Calif., Chicago, Ill., 39, 22P, Industrial and Science Conference Management, Industrial and Science Conference Management. 19-21 January 1971.

Joyner, W.H., Birman, A., Proving Simulation Between Programs, 21 October 1974, RC5091.

Joyner, W.H., Carter, W.C., Brand, D., Using Machine Descriptions in Program Verification, RC-6922

Joyner, W.H., Leeman, G.B., Carter, W.C., Automated Verification of Microprograms, April 1976, RC-5941

Joyner, W.H., Jr., Carter, W.C., Leeman, G.B., Jr., Automated Proofs of Microprogram Correctness, 9th Annual Workshop on Microprogramming, New Orleans, LA., 51-5, 27-29 September 1976.

Karnes, R.E., Carter, W.A., Computer Design Verification via Software Simulation, 74Z-00458.

Katter, O.E., Jr., Chadwick, R.S., Processor Testability Through Microdiagnostics, Proceedings of the National Aerospace Electronics Conference 1973, Dayton, Ohio, 426-33, 14-16 May 1973.

Leeman, G.B., A Method for Proving Equivalence of Programs, 10 September 1974, RC5022.

Leeman, G.B., Carter, W.C., Birman, A., Micro Program Validation: A New Aid to Computer Design Verification.

Leeman, G.B., Some Problems in Certifying Micro Programs, 5 May 1979, IEEE Transactions on Computers.

Leeman, G.B., Micro Program Certification: The Extended S-Machine Experiment, 4 December 1973, RC4640.

McCaskill, R., Wring Out 4-Bit Mup Slices with Algorithmic Pattern Generation, ELECTRON, Des., Vol. 25, No.10. 74-7. 2.

McDonnell-Douglas Astronautics Co., Houston, Tex., Simulation Verification Techniques Study: Simulation Self-Test Hardware Design and Techniques Report, 1974, 487P.

Patterson, C., Verification of Microprograms.

Oakley, J.D., Symbolic Execution of Formal Machine Descriptions, April 1979, CMU-CS-79-117, Carnegie Mellon University, Pittsburg, PA.

Ramamoorthy, C.V., Chang, L.C., System Modeling and Testing Procedures for Microdiagnostics, IEEE Trans. Comput., Vol. G-21, No.11. 1169-83.

Ramamoorthy, C.V., Shankar, K.S., Automatic Testing for the

Correctness and Equivalence of Loopfree Microprograms IEEE Trans. Comput. Vol. C-23, No.8. 768-82. 15

Reifer, Donald J., Microprogram Verification and Validation, Aerospace Corp. El Segundo California Development Operations

Robach, C., Saucier, G., Lebrun, J., Processor Testability and Design Consequences, IEEE Trans. Comput., Vol.C-25, No.6. 645-52. 12.

Robach, C., Saucier, G., Dynamic Testing of Control Units, IEEE Trans. Comput., Vol.C-27, No.7. 617-23. 13.

Robach, C., Saucier, G., Effective Test Methods for Central Processing Units, 1975 International Symposium on Fault-Tolerant Computing, Digest of Papers, Paris, France, 176-81,18-20 June 1975.

Rose, C.W., Design Systems-A Five Year View, 12th IEEE Computers Society International Conference on Computers the Next 5 Years, (Digest of Papers), San Francisco, Calif., 190-3, 24-26 February 1976.

Schoonmaker, P.B., Wenglinski, T.H., Simulation Verification Techniques Study

Schwomeyer, W.A., Verification of a Virtual Storage Architecture on a Microprogrammed Computer, AFIPS Conference Proceedings Vol.42 1973 National Computer Composition and Exposition, 401-6, 4-8 June 1973.

Stolov, S.E., Gaelic Grumman Aerospace Engineering Language for Instructional Checkout, ASSC 72 Symposium Record, Philadelphia, PA., New York, 166-72, 13-15 November 1972.

Varney, R.C., Groundwater, N.E. Criteria for Architecture Verification, 8/78 IEEE.

Wagner, Todd Jeffry, Hardware Verification Stanford University, California, Department of Computer Science.

Walter, R.H., Publications Chairman, AUTOTESTCON 79 International Automatic Testing Conference, September 1979, IEEE Publishing Services, New York, NY.

AN/AYK-15a Digital Processor Prime Item Development Specification, 16 February 1979.

AN/AYK-15A Acceptance Test Program User's Manual.

AN/AYK-15A Digital Processor, Part I, Type B1, 1 April 1979.

AN/AYK-15A Product Specification, Part II, 10 August 1979.

ISP Primer for the Architecture Research Facility, The Naval Research Laboratory, January 1976.

Symbolic Validation Algorithm Equivalence System Manual.

DATE

ILMED

8